



KING SAUD UNIVERSITY
COLLEGE OF COMPUTER AND INFORMATION SCIENCES
INFORMATION TECHNOLOGY DEPARTMENT

AraDict
*An Add-In Speech Recognition Application
to MS Word 2007*

Reference Manual

Project Submitted in Partial Fulfilment of
the Degree of Bachelors of Science in Information Technology

Submitted by:

Aciel Eshki	423200562
Dhefah Radain	423201940
Haifa AL-Thonayan	423201841
Khloud Zain-Alabdeen	423201907
Mariam Nouh	423200776
Shada AL-Salamah	423201863
Shroug AL-Megren	423202087

Supervised by:
Dr. Afshan Jafri

2nd Semester 1427\1428 (2006\2007)

Dedications

“Dedicated to my unborn child;
may your young life thrive with beauty, knowledge, and wisdom.”

-Aciel

“To my father and mother
You are truly my muse!”

-Shada

“Dedicated to my loved husband; thank you for being so patient”

-Dhefaf

Acknowledgements

We would like to express our gratitude to the following people:

Dr. Afshan Jafri, for her guidance throughout the process of bringing this project to light, and for teaching us the true meaning of self-reliance and independence.

Members of our committee, Prof. Mona Mursi, Dr. Souham Meshoul, and Dr. Layla Abu Hadeed, for their constructive feedback and valuable advice.

Dr. Nadia Al-Ghremeil and Ms. Laila Khder for their endless contributions to our department.

Dr. Shurug Al-Khalifa for her warmth, consideration and patience.

We are grateful to all those who have taught us: Dr. Ehsan, Dr. Hanan, Dr. Feryal and Dr. Lilac, Ms. Sameera, Ms. Heyam, Ms. Nahla, Ms. Abeer Al-Shayee, Ms. Aziza, Ms. Abeer, and Ms. Norah.

Special thanks to Ms. Connie for her technical advice when it was particularly needed.

This project was made possible with the help of King Abdul-Aziz City for science and Technology. In particular we extend our appreciation to Dr. Mansour Al-Ghamadi for his assistance and advice, Mr Amaar AlAnazi, a researcher in KACST, who has helped us with our work and whose insight helped to solve many problems that occurred during the project, Prof. Steve Young, from Cambridge University, who was kind enough to respond to our persistent questions, Mr. Omar Bahy and Dr. Osaama Emam, from IBM research center, for their collaboration.

Table of Contents

Dedications.....	2
Acknowledgements.....	3
Table of Contents	4
1 Abstract	6
2 Introduction	7
2.1 Speech Recognition	7
2.2 Measuring the Performance of ASR	8
2.3 Factors that Affect ASR	9
2.4 Requirements for Building an ASR	10
3 Literature Review	12
4 Problem Definition	15
4.1 Why Dictation?	15
4.2 Why Arabic?	15
5 Objectives	16
5.1 AraDict	16
5.2 Scope of AraDict	16
6 System Analysis	18
7 System Design.....	25
7.1 Users Group	25
7.2 Templates Group	29
7.3 Speech Recognition Group.....	31
7.4 Help Group	33
8 Implementation	34
8.1 Building the Recognizer	34
8.1.1 The Generic Process	34
8.1.2 Strategy1: Towards a Speaker Independent System.....	53
8.1.3 Strategy2: Replacing the Wave Files	60
8.1.4 Strategy2: Towards a Speaker Dependent System	61
8.2 Running the Recognizer	63
8.2.1 Offline Recognition	63
8.2.2 Live Recognition	65
8.3 Developing the MS Add-In Application	68
8.3.1 MS Word 2007.....	68
8.3.2 Development and Integration.....	68
9 Testing.....	70

10	Future Work	74
11	Bibliography.....	75
12	References	76
13	Appendices.....	78
	Appendix A – HTK Tools.....	78
	HTK	78
	Standard Tool Options	78
	HBuild	79
	HCompV.....	81
	HCopy.....	84
	HDMan.....	87
	HERest	92
	HHed.....	99
	HLStats.....	100
	HParse	104
	HRest.....	110
	HResults	113
	HVite	119
	Appendix B – Other Tools and Resources	124
	SAAVB	124
	ATK.....	124
	BNF.....	124
	Total Recorder	125
	Roman Representations of Arabic Characters	126
	Phonetic Representation of Arabic Sounds.....	127
	Appendix C – Our Steps.....	128
	Formal Testing of Accuracy Rates	128
	Files Created Manually Before Training.....	131
	Label Files	136
	MLFs.....	137
	Pronunciation Dictionary.....	138
	Complete List of Words in AraDict Dictionary	138
	Application Functions	139
	Appendix D – External Communications Log.....	141
14	Glossary.....	145
15	Expressions of Gratitude.....	148

1 Abstract

Previous speech recognition research has concentrated mainly on European languages. Despite being the sixth most widely spoken language in the world, Arabic has been explored to a lesser extent. The aim of this project is to develop an Arabic speech recognition system after exploring similar systems involving different languages. A dictation system, based on a continuous word system, was chosen that poses a greater challenge than speech recognition applications based on isolated word systems. Phoneme-HMMs were used as a basis for training the speech recognition engine and training was performed using the Hidden Markov Model ToolKit (HTK). As an added feature of usability, the final system was integrated to MS Word 2007 as an add-in application and given the name AraDict.

In this technical report, first, a thorough introduction is given to explore the essential speech recognition concepts. The measures of the performance of SR engines are described and the factors that demonstrate why speech recognition is a difficult task are then stated. Then, the requirements of building a speech recognition engine are explained.

A brief comparison between AraDict and some existing speech recognition systems is made. We then see why this particular kind of system was chosen and briefly describe the factors that laid the scope down.

The design of the system follow the known analysis patterns of DFDs. Based on the analysis, a detailed explanation of the implementation is give. To simplify matters, implementation is viewed as three distinct parts. Building the recognizer using training is the central theme of the project. Training is thoroughly explained with HTK commands included. After seeing how the recognizer was built and how it is tested using a formal method, the two ways of operating the recognizer are contrasted. Offline recognition and live recognition were both implemented and under each's section the main findings are stated. Then a description of how the recognizer was integrated as an add-in application to MS Word 2007 is given. And finally, after a testing session is held and results are listed, a statement of future work and the system's potential for improvement and growth are stated as well.

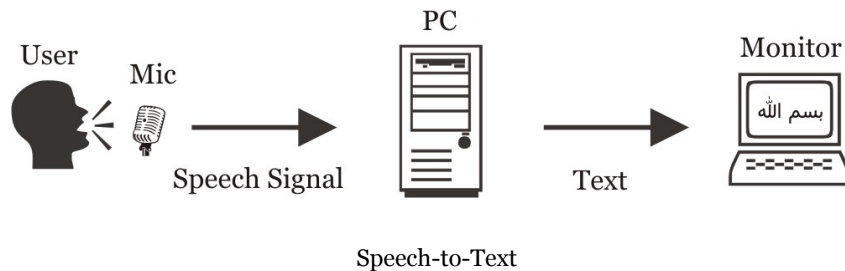
2 Introduction

2.1 Speech Recognition

Speech Recognition (SR), also known as Automatic Speech Recognition (ASR), is the process by which a computer converts an acoustic signal, captured by a microphone, into a set of words [1]. Speech recognition applications that have emerged over the last few years include voice dialing, call routing, simple data entry, content-based spoken audio search, and dictation [2]. It is however still an extremely difficult task to implement and results are not always flattering.

Dictation Systems

Dictation (speech-to-text) is a mode of speech recognition in which the user enters data by reading directly to the computer. Dictation mode allows users to dictate memos, letters, and e-mail messages, as well as to enter data using a speech recognition dictation engine [3].



Types of Speech

There are two types of speech; isolated and continuous. Isolated word systems operate on a single word at a time and require a pause between saying each word. Common isolated system applications include automatic phone answering services and speech enabled menu control systems. Continuous speech systems, on the other hand, require no pauses because they operate on words that are connected together. Dictation systems are continuous speech systems [4].

Types of Recognition

Isolated speech recognition is the simplest type of recognition because the end points of words are easy to find due to long pauses and because the pronunciation of a word is not affected by its neighbours. This also makes it the most accurate type of recognition. Continuous speech, on the other hand, is more difficult to handle because it is hard to find the start and end points for each word due to the flowing nature of the speech [1].

Speaker Dependent vs. Speaker Independent Systems

A speaker dependent system is developed to operate for a single speaker whereas a speaker independent system is developed to operate for any given speaker of a particular type, for instance, Saudi accented. The former produces higher accuracy rates while the latter presents the user with more flexibility. Speaker independent systems however are not completely independent; they require a speaker adaptation session to train the system to accept a new speaker of the particular type. These adaptation sessions are nevertheless shorter than those required to train a new speaker for a speaker independent system.

2.2 Measuring the Performance of ASR

Accuracy

The performance of speech recognition systems is typically measured in terms of the accuracy rate, which is an approximation of the percentage of *correctly* recognized words. Another representation of accuracy is the word error rate (WER) which as its name suggests represents an approximation of *incorrectly* recognized words. For example, if a system can achieve up to 70% accuracy, we say it has a WER of 30%.

Size of vocabulary

A speech recognition system is limited by the size of its vocabulary or the number of words it was designed to recognize. These words are typically predefined in the system's dictionary. Any words that are not included cannot be recognized, and are referred to as out-of-vocabulary words (OOV). Although increasing the size of vocabulary might seem like a good idea, we find that it decreases the accuracy rate significantly. The challenge is to try to increase the amount of vocabulary while maintaining a high level of accuracy rate.

Speed

Speed is measured in terms of the real time factor (RTF) or the ability for the system to respond in a real time manner. Speed can also be measured in terms of latency, which is a time delay between the moment speech is initiated by the speaker and the moment results becomes detectable [5]. However, because no formal methods for calculating the latency of the speech recognition system were found, we informally expressed the speed of our system in terms of instant response vs. presence of some delay.

2.3 Factors that Affect ASR

Spoken language involves several factors that affect the speech signal and make it difficult for machines to recognize speech.

1. Coarticulation of Continuous Speech

We normally speak by producing a continuous, connected stream of sounds that fit closely together with each other. Phonetics tends to view speech as a sequence of segments, called phonemes, where each phoneme represents a distinct sound. However, to imagine these segments as discrete and independent of each other would be quite wrong [6]. In spoken languages, we find that phonemes have strong impact on neighboring phonemes. This impact results in modifying their original sounds. Continuous speech takes this a step further by allowing phonemes in one word to affect phonemes in neighboring words, as there are no clear boundaries between words. This process is referred to as coarticulation. As an example, when we say (في المساء), we omit the sounds of the “ya” at the end of (في) and the “alef” at the beginning of (المساء), which results in something that sounds like (فيلمساء).

2. Variations in Speech

The variation of human voices is immense and we have to try to cope with all kinds of speakers that utter the same thing in acoustically distinct ways. Even the same speaker varies acoustically depending on physical and emotional factors such as having a cold or being stressed. Individual variations are not only on the voice level but speakers also behave different in dialogues and have different talking habits [7].

3. Environmental Variations

In addition, a recognizer has to be able to distinguish speech from other acoustic signals such as noise. If in case the acoustic environment was held constant, meaning that the disturbance was relatively stable, then this could be modeled using a feature extraction algorithm. Feature extraction tends to automatically estimate the signal-to-noise ratio (SNR) between speech and noise. But more often that not, we find systems being exposed to many different environments which are hard to handle, resulting in inaccurate recognition [7].

4. Disfluencies in Speech

Apart from these factors, people do not speak as clearly and eloquently as they think they do, but produce filled pauses, repetitions, repairs, utter truncated words, make false

starts, make mistakes and slips of tongue and even change their minds during speech production. How shall we cope with all these disfluencies? We also produce a lot of extralinguistic sounds such as inhalations or coughs. Imagine the difficulty to make a machine distinguish between linguistic and extralinguistic signals! A small comfort is that users shape up a little bit in dialogues with machines and speak clearer than they should have done with a human dialogue counterpart. The occurrence of disfluencies is still common though in human-machine dialogues but are less frequent than in human-human dialogues [7].

5. Vocabulary vs. Accuracy rate

Another aspect is the problem of vocabulary. People will always come up with words that developers had not thought of i.e. out of vocabulary words (OOV). How shall we handle unknown words? We could of course propagate for larger vocabularies, but a larger vocabulary also complicates the recognition task [7], as the more words to distinguish from, the higher the probability of acoustically similar words to confuse the input word with. This inevitably causes the word accuracy rate to significantly decrease.

6. Human vs. Machine Interpretation of Speech

Humans can often instinctively compensate for all of these disturbing factors on the speech signal whereas today's speech recognizers cannot. Humans in contrast to automatic recognizers easily distinguish speech from noise, can recognize unknown words and understand incorrect utterances with help of contextual interpretations [7].

2.4 Requirements for Building an ASR

HMMs

There are many possible variations in speech signal and visually similar waveforms do not necessarily indicate similar sounds. Thus, performing pattern recognition algorithms on the speech signal is not an efficient way to recognize speech.

The most widely used approach for implementing a speech recognition system is a statistical framework called Hidden Markov Models (HMMs). Their significance lies within their ability to simulate the structure and form of a given system, which in our case is the Arabic language. This is done by processing a speech corpus (which would be the basis for defining their structure) using sophisticated tools. The process of defining the structure of the HMMs is called training, which is a lengthy and complex process.

Phoneme-Based HMMs

We chose a phoneme-based HMM system; meaning that each HMM will represent a single phoneme. In human language, a phoneme is the theoretical representation of a sound. It is a sound of a language as represented (or imagined) without reference to its position in a word or phrase. A phoneme, therefore, is the conception of a sound in the most neutral form possible. Phonemes are not the physical segments themselves, but mental abstractions based on what a speaker of a language thinks of, hears or sees as being acoustically the same [8].

We take for example the Arabic language; Arabic consists of 28 alphabets, each having a distinct sound. One phonetic representation of the Arabic language can be a single phoneme for each alphabet, making it 28 phonemes in total. Another representation can consider the three long vowels (مد الألف، مد الواو، مد الياء) as additional phonemes making it 31 phonemes. A final representation considers also the short vowels, represented by the three diacritics (فتحة، ضمة، كسرة) as additional phonemes, making it 34 phonemes.

It is essential to point out that phonemes are not equivalent to syllables. While a phoneme is an abstract representation of a single sound, a syllable is physical unit of organization for a sequence of speech sounds.

Speech Corpus

A speech corpus is a collection of recorded utterances and their associated transcriptions used as a basis for the descriptive analysis of a language. The transcriptions are represented by the written form of the language.

Training Tools

Training defines the structure of HMMs. It requires a set of tools that perform sophisticated internal algorithms given a set of specific parameters. These algorithms are based on complex mathematical and statistical knowledge.

The Hidden Markov Model Toolkit (HTK) is a general purpose toolkit developed by Cambridge University Engineering Department. It is used for numerous purposes one of which is training speech recognition engines. It runs with command line style interface.

3 Literature Review

Work on speech analysis began in the early 70s under the leadership of Professor Frank Fallside. Dr Steve Young (now Professor Young) started working more specifically on speech recognition at Cambridge University Engineering Department (CUED) in the late eighties. A major breakthrough was made in the early 90s in the methodology of acoustic modelling and the development of HTK, which is a toolkit, designed to train acoustic models for ASR system. This put the research of the Cambridge team in the forefront of speech recognition systems worldwide.

The HTK system effectively demonstrated that desktop dictation was possible and companies like Dragon and IBM subsequently converted these ideas into commercial products. The CUED team continued to improve their system tackling harder tasks such as dictation in noise, and most recently transcription of broadcast news material. The latter is particularly difficult because the recognizer must cope with a sequence of unknown speakers, speaking over different channels with varying degrees of background noise.

Although much has been achieved over the last decade, speech recognition technology is still very limited and new research continues apace [9].

Arabic ASR Systems

The most prominent work in Arabic dictation was IBM's ViaVoice systems, which have been discontinued; no reasons for the discontinuation were disclosed. Other systems such as BBN Arabic CallHome produce noticeably low accuracy rates; 54.5% WER, compared to their versions in other languages, BBN English CallHome; 30% WER. Due to the lack of milestones in Arabic ASR, there are very few standardized resources to base our work upon which presented us with a greater challenge.

English ASR Systems

Speech recognition systems implemented in English are far more successful than those implemented in Arabic. In order to produce a higher quality Arabic dictation system it is essential to examine a successful English ASR system. The most outstanding dictation system is the commercial speech recognition package Dragon NaturallySpeaking 8. In the following table we compare it to our system:

	Dragon NaturallySpeaking 8	AraDict
Language	Five English dialects: US English UK English Australian English Indian English Southeast Asian English	Standard Arabic
Functional Specifications	Dictation Automatic insertion of punctuation into text (ex: new line, periods, commas, new paragraph) Menu control	Dictation; considers only text and a single cell-skipping command
Dictation Mode	Free dictation	Blank documents and application specific templates
Scope of Application	All Windows-based applications	MS Word 2007
System Type	Commercial	Experimental
Context	Various areas	Administrative
Mouse and Keyboard Usage while using speech	Enabled	Enabled
Speaker Dependency	Speaker independent; recommends training sessions (speaker adaptation) for every new speaker	Speaker dependent; requires a single dynamic training session for every new speaker
Training Period Length	Varies from 10 minutes to a couple of hours; the more the system is trained the better the accuracy rate	15-20 minutes
Speech Type Used	Isolated and continuous	Continuous
Text And Speech	Supports speech-to-text and text-to-speech	Supports speech-to-text only

	Dragon NaturallySpeaking 8	AraDict
Number of Words that can be Recognized	Not specified	Approximately 50 words
Adding New Words by User	Possible	Not possible
Interface Menu	Toolbar interface	Word 2007 ribbon interface

4 Problem Definition

4.1 Why Dictation?

Wrist Injuries and User Disabilities

For frequent keyboard users, there is a potential risk of developing typing related injuries in the long run. Dictation systems decrease this potential risk by allowing the creation of documents using a microphone while minimizing the use of the keyboard. Many computer users also have disabilities that limit the way they use the computer. Those who are unable to type, due to a broken arm or impaired vision, for instance, can make use of dictation systems that conveniently allow hands-free document creation.

A Greater Challenge

Isolated speech recognition systems, such as speech-enabled dialing, have been implemented with a greater degree of success than continuous ASR applications such as dictation, which presents us with a greater challenge.

4.2 Why Arabic?

Significance of the Arabic Language

Besides being our mother language, Arabic is currently the sixth most widely spoken language in the world with an estimated number of 250 million speakers [10]. Despite this fact, there has been little research on Arabic speech recognition compared to other languages of similar importance (e.g. Spanish or Mandarin).

Language Inherited Difficulties

Previous research on automatic speech recognition (ASR) has mainly concentrated on European and Asian languages. Semitic languages such as Arabic and Hebrew have been explored to a lesser extent. These languages possess certain characteristics which present problems for standard ASR systems. One problem is that their written representation does not contain most of the vowels present in the spoken form. For example, the pronunciation of the word (كتب) cannot be determined due to the absence of short vowels; there is no way to tell if it is pronounced “kataba” or “kutub.” One way to resolve this is to enforce diacritics to compensate for the absence of vowels in the written form of the language.

5 Objectives

5.1 AraDict

As its name suggests, AraDict is an Arabic dictation system, which converts continuous Arabic speech to text. Rather than implementing it as a stand alone text editing application, we chose to implement it as an add-in application to MS Word 2007. MS Word is already the most widely used word processor for PC users and the features it offers are limitless, which presents the user with a higher degree of usability for our system. And being the most recent edition, MS Word 2007 was our choice.

5.2 Scope of AraDict

Modifying the Initial Scope

Initially, we were aiming to develop AraDict as a speaker independent system which can be used to dictate freely up to 300 words. However, due a lot of limitations on the speech corpus, and due to challenges faced during implementation ([For details, see Implementation](#)), we decided to modify the initial scope making AraDict application-specific.

Speaker Dependence

Creating a speaker independent system was infeasible due to incompatibility issues faced with the corpus, as we will see in [Implementation](#). As an alternative, we implemented a speaker dependent system, but equipped it with the ability of adding a new user.

Size of Vocabulary and Usability

Another modification made to the scope was decreasing the size of vocabulary. The only speech corpus we could find was the Saudi Accented Arabic Voice Bank (SAAVB), which supplied us with a maximum of 250 words. However, because SAAVB was constructed for a telephone-based system, the words it included were insufficient for dictation due to their unrelated nature and lack of constructive meaning. Their unsuitability for free dictation led us to construct application-specific word templates of administrative nature and facilitate filling them easily using dictation.

These templates are made up of formal tables, forms, and schedules that are used regularly by KSU administration. They required only a limited number of words, and so we tremendously decreased the size of vocabulary without robbing the system of its usability. AraDict, with a vocabulary size of approximately 50 words, can now be used to

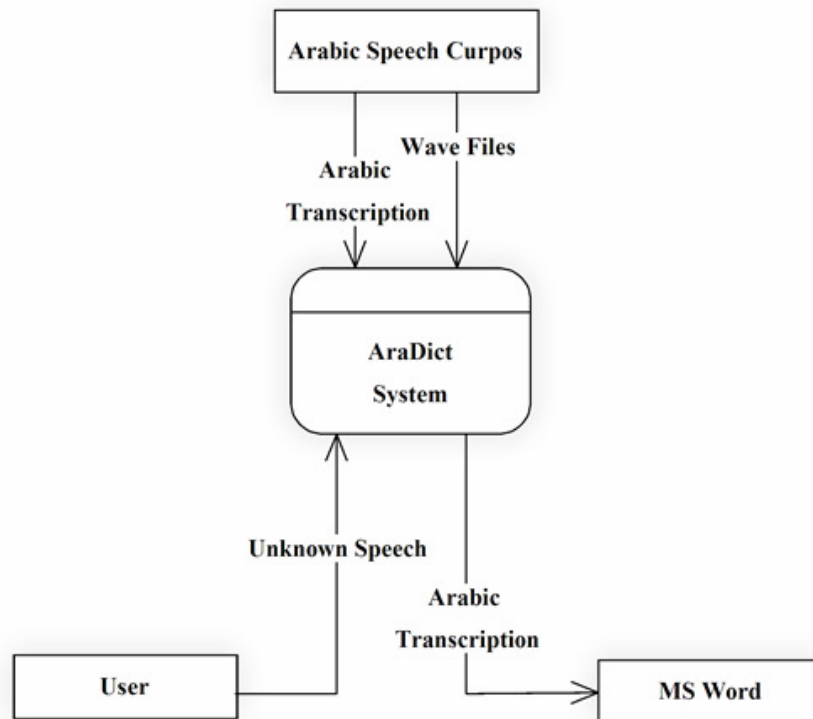
completely dictate three different templates with numbers, courses, and names of people, and a single cell-skipping command. For a complete list of vocabulary see [Appendix C – Our Steps](#).

Experimenting with Recognition Methods

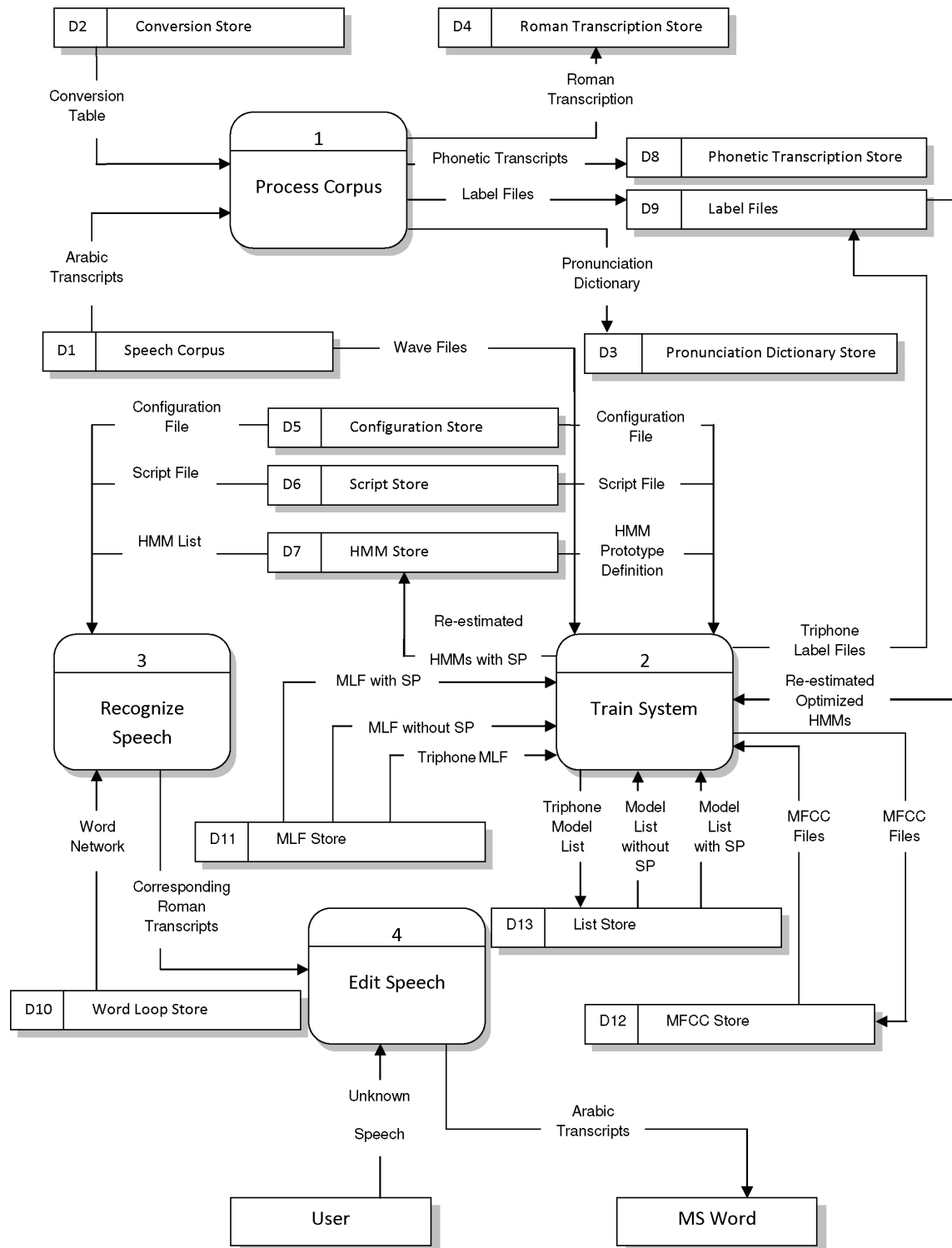
After exploring the two different methods for operating the recognizer we decided to implement both in an experimental sense. Live recognition proved difficult, inconsistent, and unpredictable as appose to recoded recognition (also known as offline recognition) which produced very good results. AraDict facilitates both methods even though the option adds little value to the average user. We hope to improve live recognition and make it real time in the near future ([see Future Work](#)).

6 System Analysis

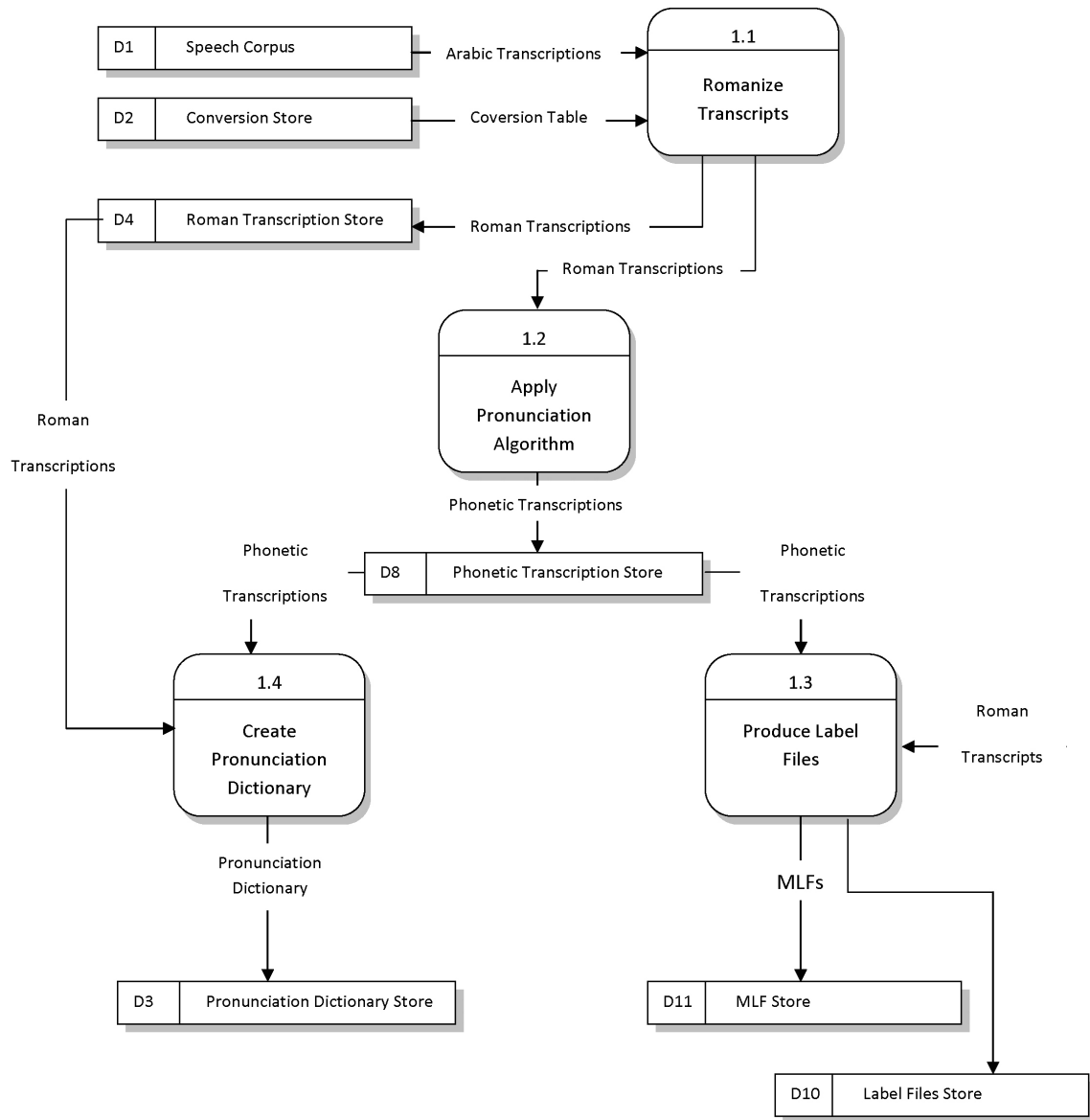
After observing the analysis of existing speech recognition systems, we decided to express our analysis using simple Data Flow Diagrams (DFDs). The analysis describes the process of implementing AraDict and the details are thoroughly explained in [Implementation](#).



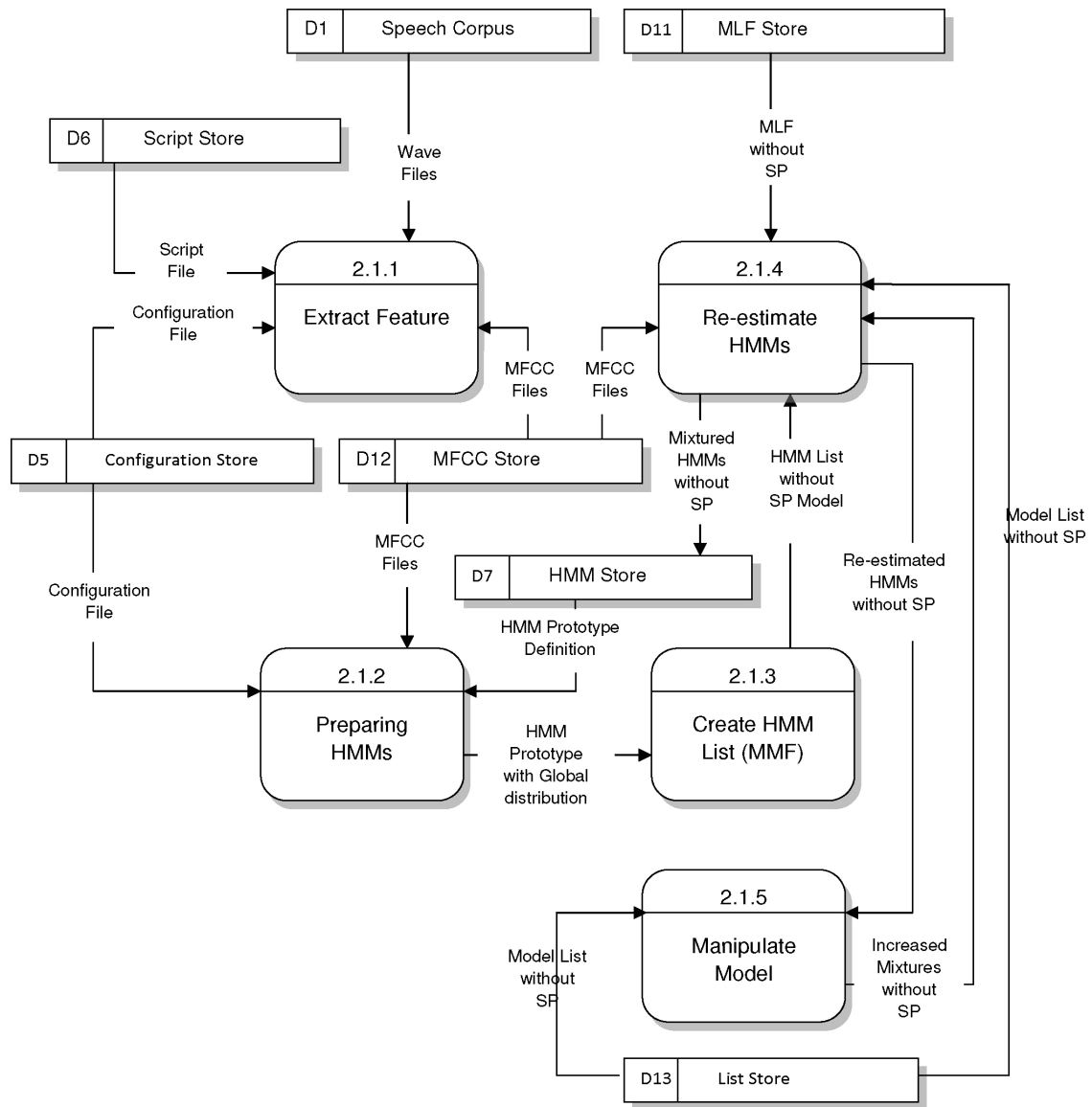
Context Diagram



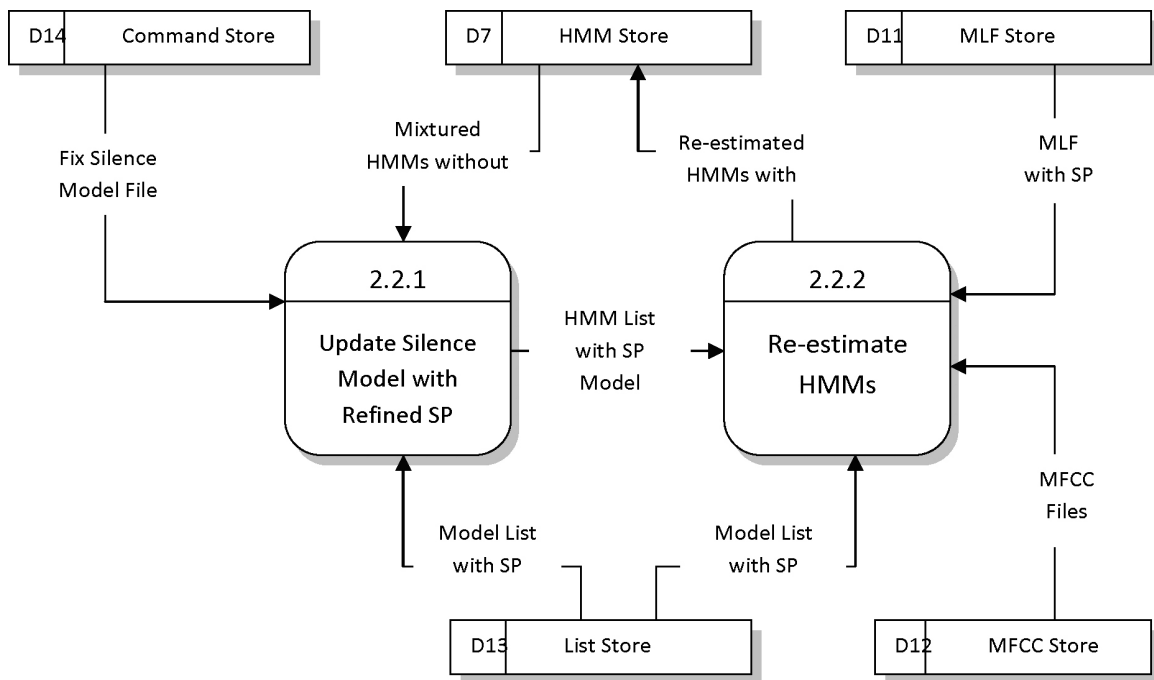
Level-o Diagram



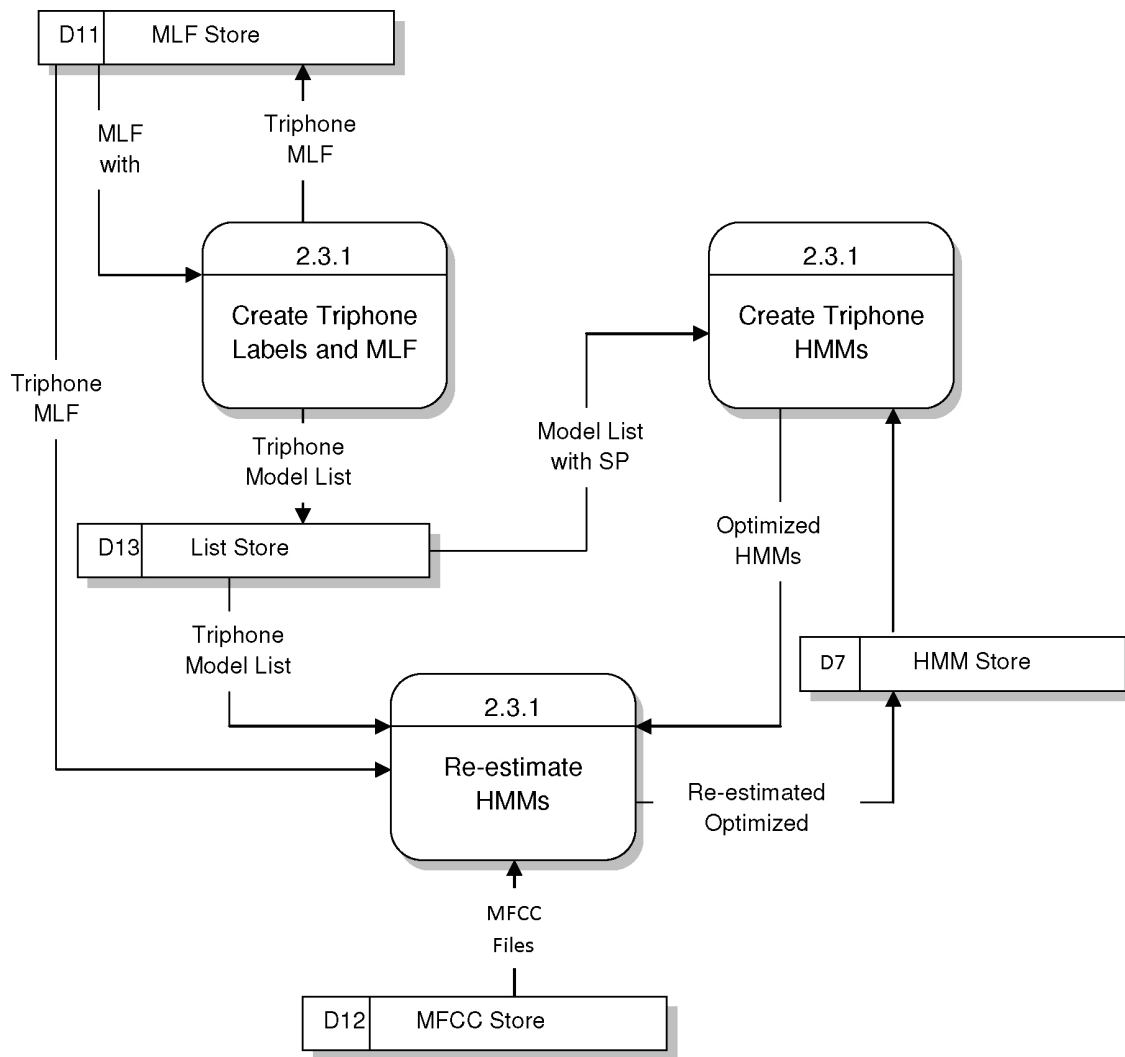
Level-1 for Process 1: Process Corpus



Level-2 for Process 2.1: Train without Short Pauses



Level-2 for Process 2.2: Train with Short Pauses

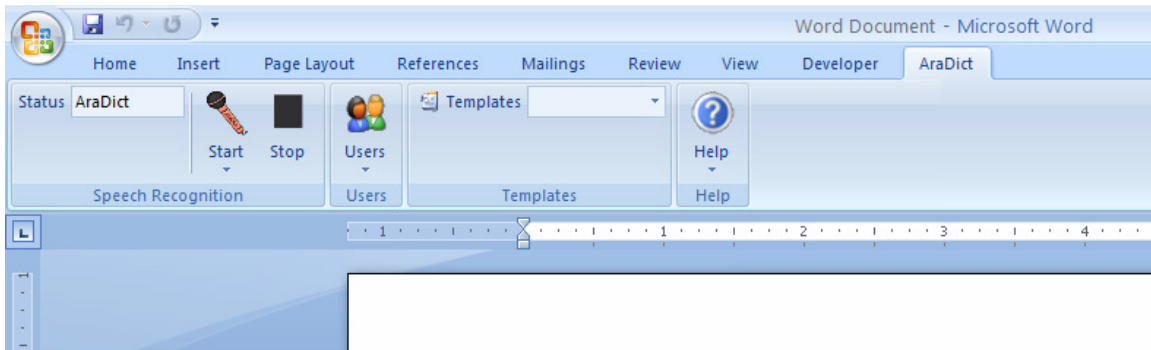


Level-2 for Process 2.3: Optimize using Tri-phones

7 System Design

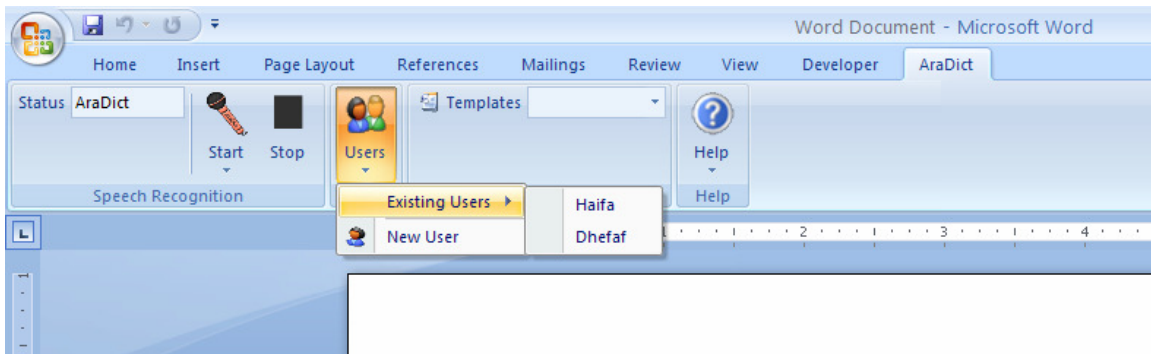
The Interface

After installing AraDict, it will appear under a tab automatically when MS Word 2007 is started. When the tab is clicked, the AraDict Ribbon will appear. The ribbon contains four groups: the user group, the templates group, the speech recognition group and the help group.



7.1 Users Group

Because AraDict is a speaker dependent system, each individual user must own a user account. The users group maintains these accounts.

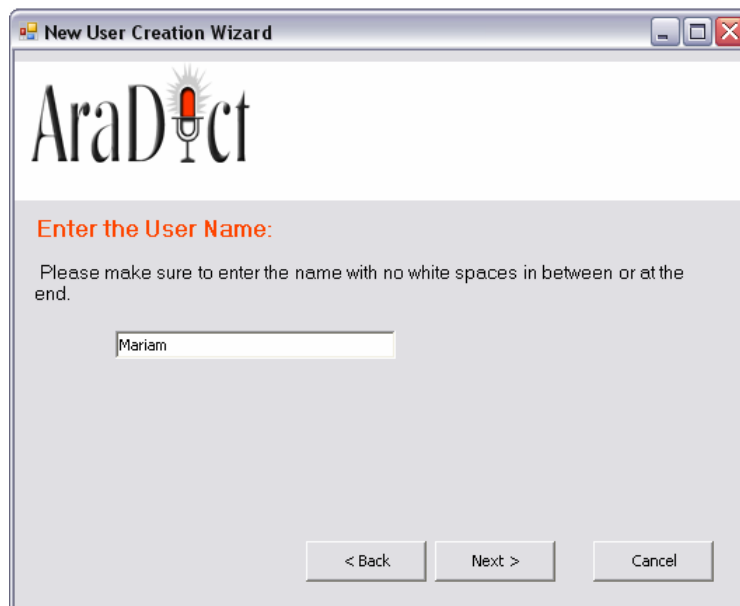


Creating a New User

For a new user to start using AraDict, he or she must first create an account. The New User Wizard is initiated by clicking on “New User” in the Users drop-down menu. The following wizard window pops and the creation process begins.



1. The first step is to enter the user name. The name must be unique; otherwise it will overwrite an existing user. Names are not case sensitive and must contain no spaces.

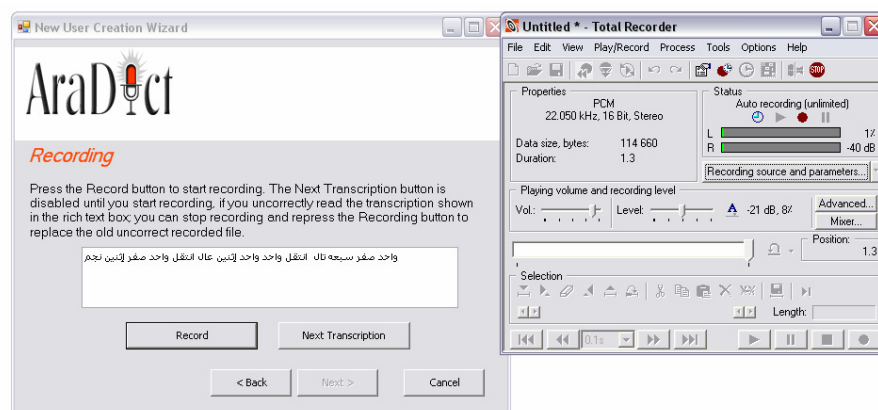


2. The user is then prompted to start recording all the transcriptions in the dictionary, which include the three categories: numbers (1, 2, 3...), names of courses (عرب، تال، عال...), and names of people (علي، شروق، عمر...), plus a single cell-skipping command (انتقل). The "Next Transcription" button is disabled until the user

starts recording. If the transcriptions were incorrectly read; the user can stop recording, press “Record” button again, and re-read the transcriptions.



3. When the user presses the “Record” button, a recording tool, Total Recorder, pops up and the user can start speaking with his or her own natural flow. Each recording last no longer than a minute, and the entire recording session takes from 15-20 minutes maximum. When the user is done recording, he or she must click on the stop button on Total Recorder before closing the window, in order to save the recording. “Next Transcription” is then enabled and the user can click on it to move to the next transcription.



4. When all the transcriptions are recorded, the wizard shows a message box indicating that the recording session has ended.

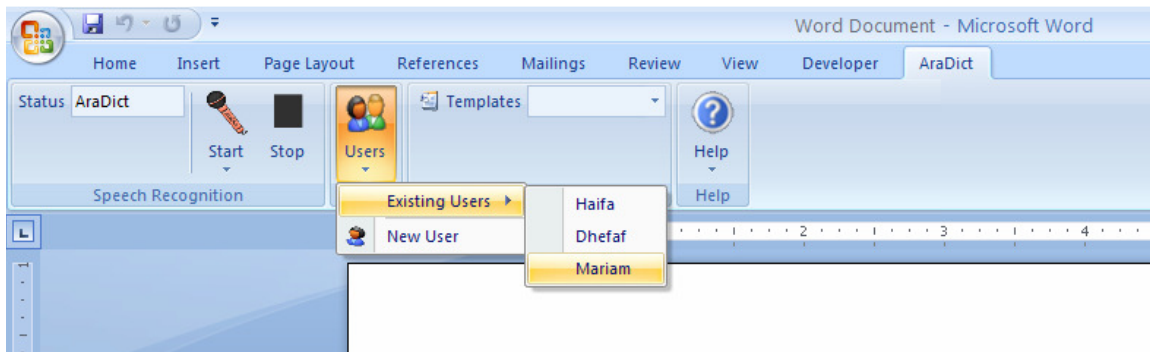


5. The user can then start training the system by clicking on the Start Training button. When the progress bar stops moving, the system indicates that the training step was done successfully. Then the user can press the Next button to finish the creation of the new user.





The new user is then inserted into the Users Group under “Existing Users.” If the user did not appear immediately after training, Word must be closed and reopened.



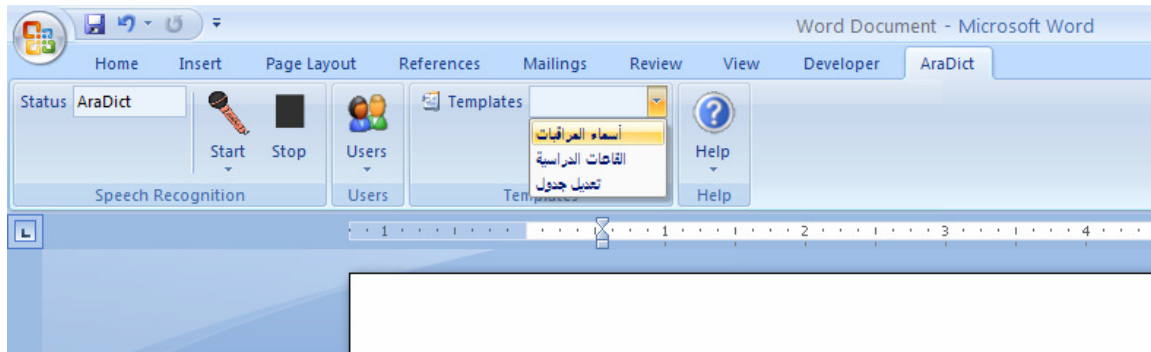
7.2 Templates Group

Dictating a Blank Document

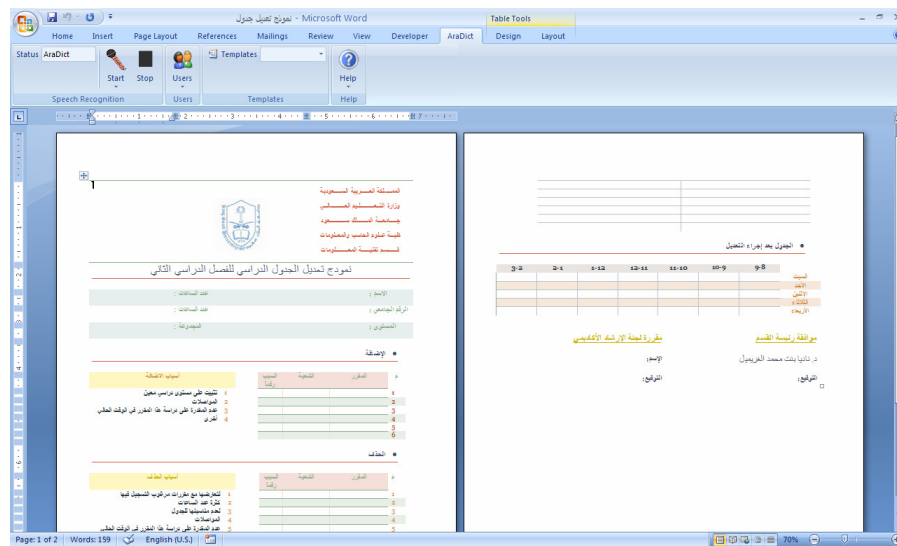
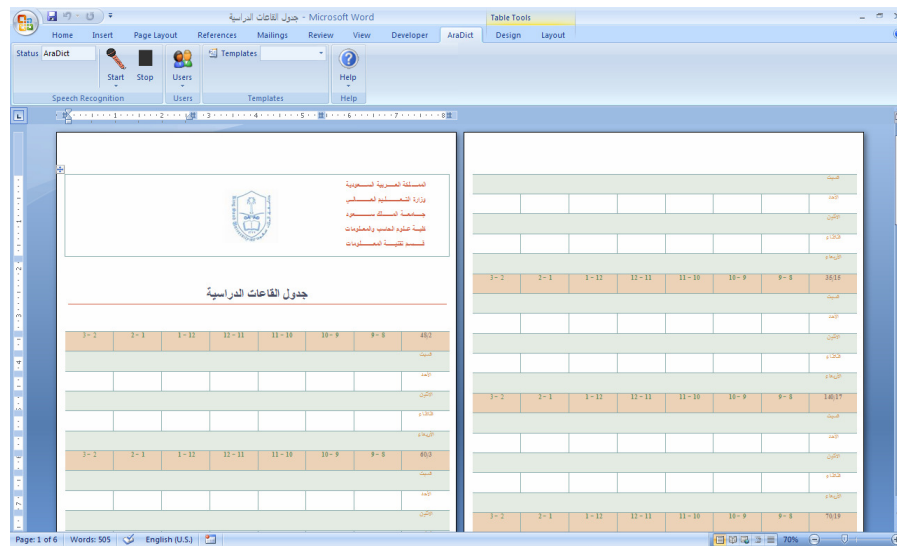
The user has the choice to either dictate a blank document freely using the limited vocabulary provided with the system, or use one of the provided templates. When dictating a blank document, the single cell-skipping command (انتقل) is disabled and instead it is typed as any other word in the dictionary.

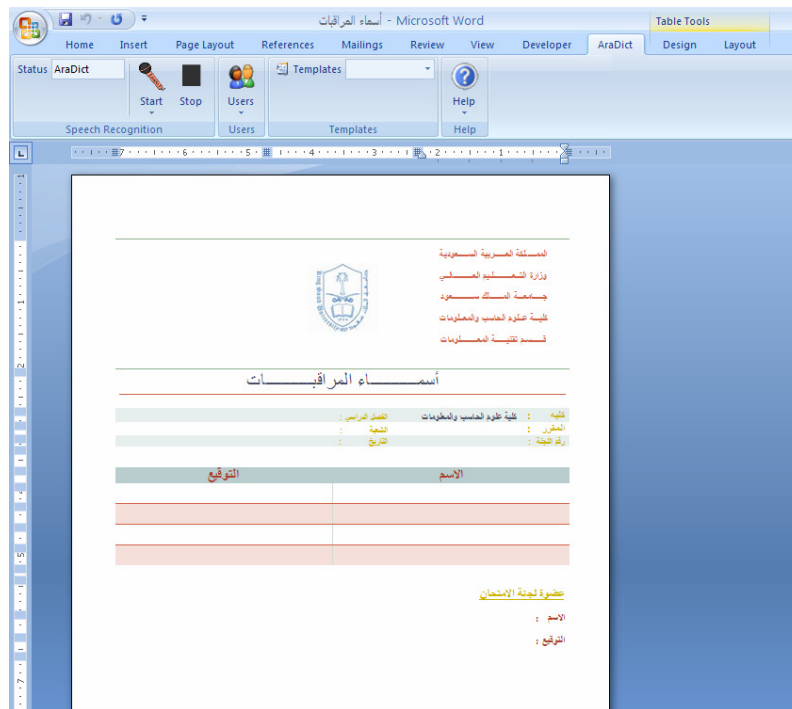
Dictating a Template

Three templates were developed, as shown in the images below.



All three templates are of administrative nature. They are made up of formal tables, forms, and schedules that are used regularly by KSU administration.

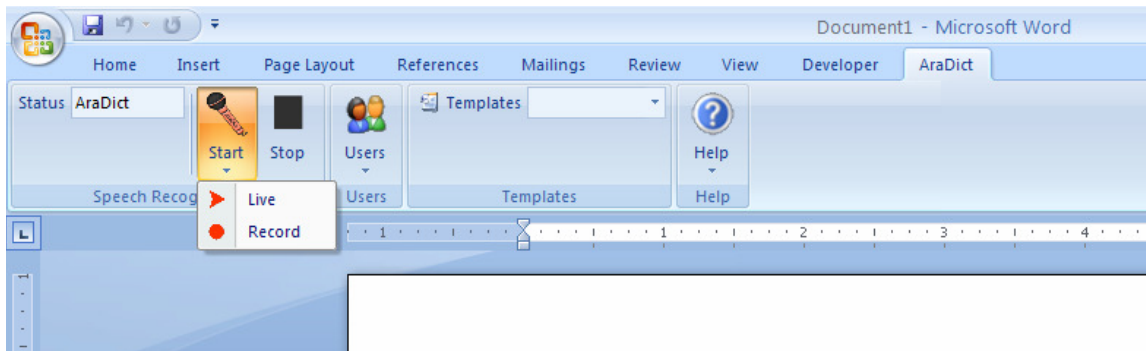




Dictation facilitates filling them easily using speech. A single cell-skipping command (انتقل) is provided for a complete hands-free experience.

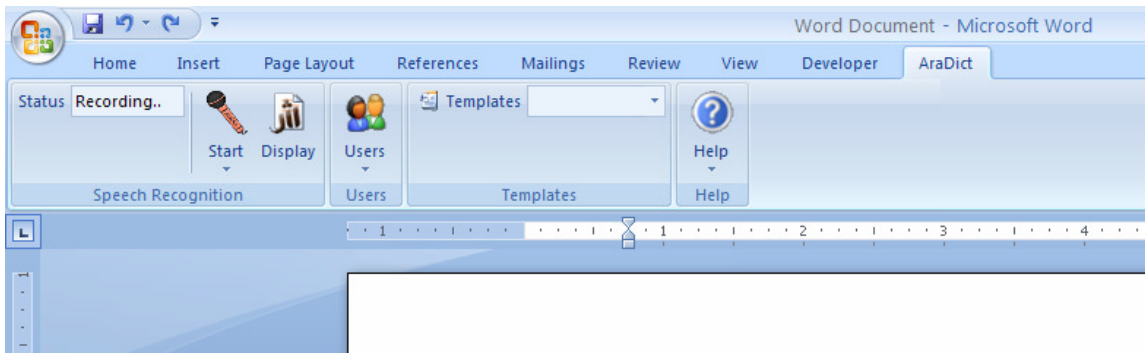
7.3 Speech Recognition Group

The speech recognition group is where the actual speech recognition takes place. It contains three elements: a “stat” button, a “stop” button” and a “status” bar. The status bar shows the current mode the user is operating on. Two modes exist from which the user can select: live and recorded. The recorded is the one recommended. It is absolutely important for a user to be selected before selecting a method.



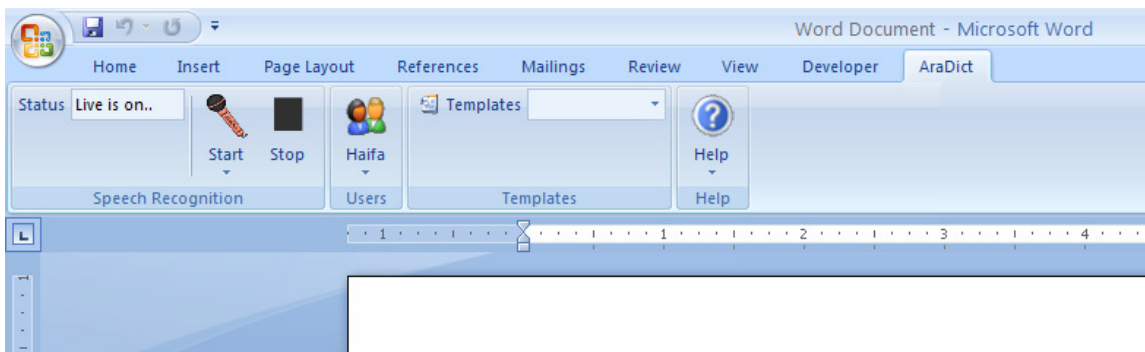
Recorded Recognition

When the user presses the “Record” button, the current “status” changes to “Recording...” and the “stop” button is converted to a Display button. The system will pop up a recording tool, Total Recorder, and the user can start speaking with the same natural flow he or she used during training. The recording should last no longer than a minute and the stop button on Total Recorder should be pressed before closing the recorder window in order to save the recording. The user is then required to press the “display” button in order to display the text on the document or template. Pressing the Display button changes the systems status to off by displaying the “OFF...” Message



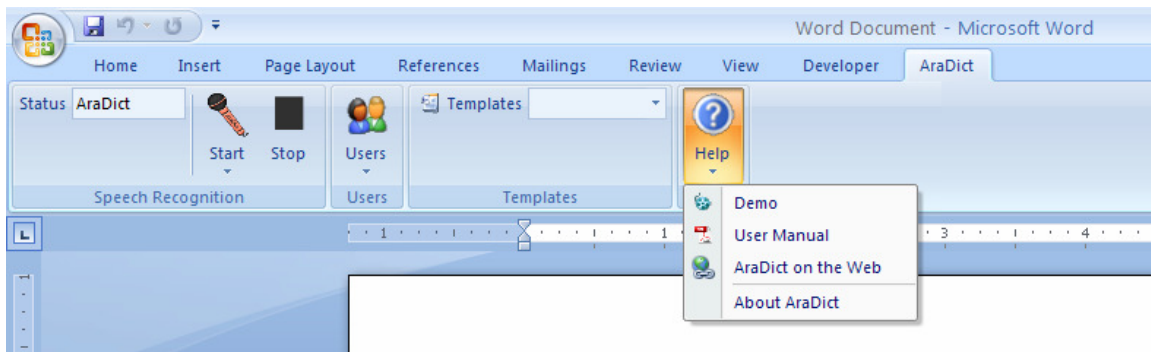
Live Recognition

When the user presses the “Live” button the current “status” changes to “Live is on...” and the “stop” button returns to its original state. The user can start speaking with the same natural flow used during training. When the user is done, he or she should click on the “stop” button on AraDICT in order to display the text on the document or template. Pressing the “stop” button changes the systems status to off by displaying “OFF....”



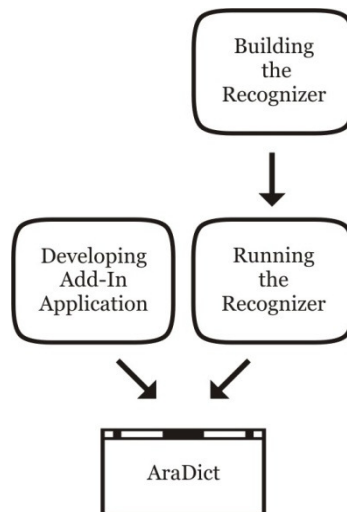
7.4 Help Group

Aradict Demo, User Manual, AraDict on the Web and About aradict are provided in the Help group.



8 Implementation

We can view the implementation of the system as three distinct parts. We will first explain the central part of implementing the system which is building the recognizer. We will then examine the different methods for running the recognizer and how they were implemented. Finally, we will explain how the add-in application to MS Word 2007 was developed, and how the recognizer was integrated into it making it a complete ASR application.



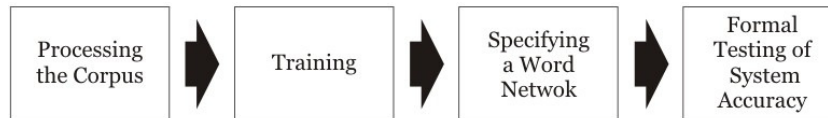
The process

8.1 Building the Recognizer

Building the recognizer was the most essential step in implementing the system. Initially, we adapted a strategy of building a speaker independent dictation system. This strategy, however, failed towards the end for reasons that are disclosed, which led us to consider two other alternatives. The first alternative produced poor results, while the second was successful. Each of these strategies will be explained individually, but first, we will explain the steps which are common to all three. We will refer to these steps as “the generic process.”

8.1.1 The Generic Process

Training is the central theme of building any speech recognizer. For this training process a speech corpus must be available. The corpus is first processed before being submitted to the extensive training phase. After the training phase is complete, a word network is specified. The recognizer is then ready to be tested.



The Generic Process

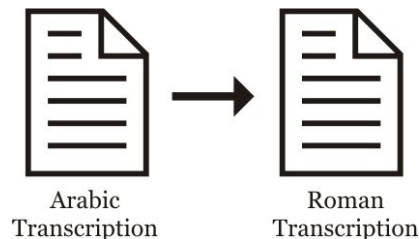
Stage 1: Processing the Corpus

This is a pre-training stage where data from the corpus is processed and prepared to match the format required for training using the HTK tools. In order to automate as much of the process as possible, several algorithms and small programs were written. A few files, however, were manually created.

Romanization of Arabic Transcriptions

Throughout the speech recognition process, HTK will be used. Several HTK tools receive text files as input. These text files must only contain characters that the HTK tools can identify with and accept. HTK tools process textual data in roman form (A, a, B, b...etc.), and thus Arabic characters needed to be transformed into a roman equivalent. We chose a common roman representation ([Appendix B – Other Tools and Resources](#)) which substitutes each Arabic character (letters and diacritics) with a roman character.

Any roman character can replace any Arabic character with no regard to sound. It is a mere symbolic representation and sounds produced by the characters have no significance. For example, we can replace the character (ﺏ) with the character (Z), and the character (ﻑ) with the character (f) because the sound of the characters are irrelevant. However, it is a good idea to be consistent. After choosing the common representation described in the appendix, we wrote a simple mapping function to automate the substitution process for Arabic transcription files.



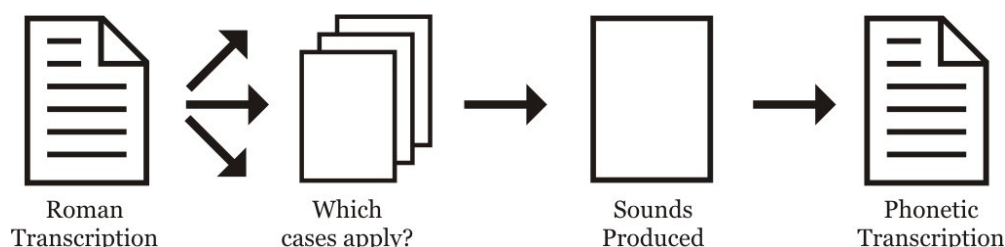
Romanization of Arabic Characters

Pronunciation Algorithm

The pronunciation algorithm deals with converting text into a sequence of phonemes that describe the way this text is pronounced. We chose a common phonetic representation

that substitutes every possible sound in the Arabic language with a phonetic symbol ([Appendix B – Other Tools and Resources](#)). This is also a symbolic representation that does not require the substitutes to match in sound. Any Arabic phoneme can be replaced with any phonetic symbol. Phonetic symbols are usually denoted with two lower case roman characters.

Unlike the previous step, the pronunciation algorithm is not a straight forward mapping function but rather a more complex algorithm. This is due to the fact that same letters are pronounced differently depending on the context in which they appear. The algorithm must therefore consider not only the cases where the written representations match the spoken form, but also the special cases that produce sounds that are different from the written representation.



Pronunciation Algorithm

A good example of the mismatch of spoken and written representations is (ال التعريف). The (ل) is either pronounced naturally (*Qamariya*) or omitted (*Shamsiya*) depending on the letter that follows it. In cases where it is omitted, the letter that follows is emphasized with a (Shadda). Also, when the word containing (ال التعريف) occurs in the middle of the sentence, the pronunciation of the letter (ل) is also omitted, as appose being pronounced when occurring at the beginning of the sentence.

A second case is (همزة الوصل) which is also either omitted or pronounced depending on its location in the sentence.

A third important example concerns the three letters (ألف، واو، ياء). The dilemma lies within knowing when they are pronounced as consonants and when they are pronounced as vowels. The diacritic on the letter preceding these three characters, as well as the diacritic on the letter itself, determine whether they are vowels or not. (ألف) is preceded with a (فتحة), (واو) is preceded with a (ضمة), (ياء) is preceded with a (كسرة), and all three must have a (سكون) diacritic.

A forth case is (التاء المربوطة: ة), which can either be pronounced as a (هـ) or a (ت) depending on whether it occurs at the end of the sentence or in the middle.

The final case is special words such as the word (مائة) which is pronounced (منة). There are many examples of the variations of pronunciation in the Arabic language, but only these needed to be considered for our system.

Label File Producing Functions

Many of the operations performed by HTK assume that speech is divided into parts and each part has a name or label. The set of labels associated with a speech file constitute a transcription and each transcription is stored in a separate label file.

Three types of label files were required for training, a phoneme-based with short pauses, a phoneme-based without short pauses, and finally a word-based ([Appendix C – Our Steps](#)). For each type a separate function was written.

The function written to create the word-based label files accepted the roman transcriptions as input. It simply separated each word in an individual line then output the result to a new file.

The function written to create the phoneme-based label files accepted the phonetic transcriptions as input. It separated each phone in an individual line but added an extra phone, called the short pause (sp), between each word in the first kind. The results were output to new files.

Note that for each individual wave file in the corpus all three kinds of label files will be created. This will result in a very large number of files. A good solution is to combine the set of similar files into a large file called the Master Label File.

Master Label File Producing Function

The Master Label Files (MLFs) producing function simply combined the smaller label files into a large file ([Appendix C – Our Steps](#)).

Dictionary Producing Algorithm

The dictionary is a text file that lists the possible pronunciations for each word ([Appendix C – Our Steps](#)). It is the file the recognizer uses for recognition. Words are obtained from the roman transcriptions and their pronunciations are obtained from the corresponding phonetic transcriptions. It is always a good idea to decrease the number of words in a dictionary in order to increase the accuracy of recognition.

We wrote an algorithm to generate a unique sorted listing of all possible words and their pronunciations.

Manually Created Files

See ([Appendix C – Our Steps](#)).

Stage 2: Training

An extensive training stage is required for every speech recognition system. HTK was our choice of training toolkit ([Appendix A – HTK Tools](#)). There is no single rigorous way to train ASR systems; steps and parameters vary depending on the desired outcome and usage of the final system. The HTK Book (found online at CUED) was thoroughly studied and methods of training were selected accordingly.

1. Feature Extraction

The first step in training was to perform feature extraction on the wave files. This step is essential when building a speech recognizer since speech recognition tools cannot process directly on the waveform and the wave files need to be represented in a more compact and efficient way. This is called acoustical modeling. The goals of performing feature extraction are as follows:

- Remove part of the speech signal that does not contribute to the phonetic identity
- Reduce the amount of noise introduced by the environment and the recording hardware (automatic estimates the SNR in an assumed static environment)
- Reduce the amount of data to be processed and compress it into manageable form

For speech recognition, the most common acoustic feature in use is Mel-scale Frequency Cepstral Coefficient (MFCC). MFCCs model the human auditory system responses more closely than other representations which will allow for better processing of data. For those reasons our choice on the representation of features was based. MFCCs are derived as follows:

- The signal is segmented in successive frames, overlapping each other. Each segment will be represented with a Fourier transform. This step is necessary, since it is not preferable to represent the entire signal as one Fourier transform. A Fourier transform is mathematical operation that resolves the frequency components of a particular system. It basically shows what high, medium, and low sounds and what volume of each, are combined to make a complex sound like the human voice.

- The amplitudes of these frames will be mapped onto the Mel-scale. The Mel-scale is logarithmic scale of frequency that is based on human pitch perception.
- Then a type of Fourier transform, called discrete cosine transform that is only used for real numbers, is taken of the list of Mel amplitudes. These amplitudes represent the MFCCs.

MFCCs are usually represented as a vector of acoustical coefficients that are extracted from the segmented frames.

A configuration file will be used to set the parameters for feature extraction. HTK tool, HCopy, will then use these settings to transform the wave file into the more efficient representation.

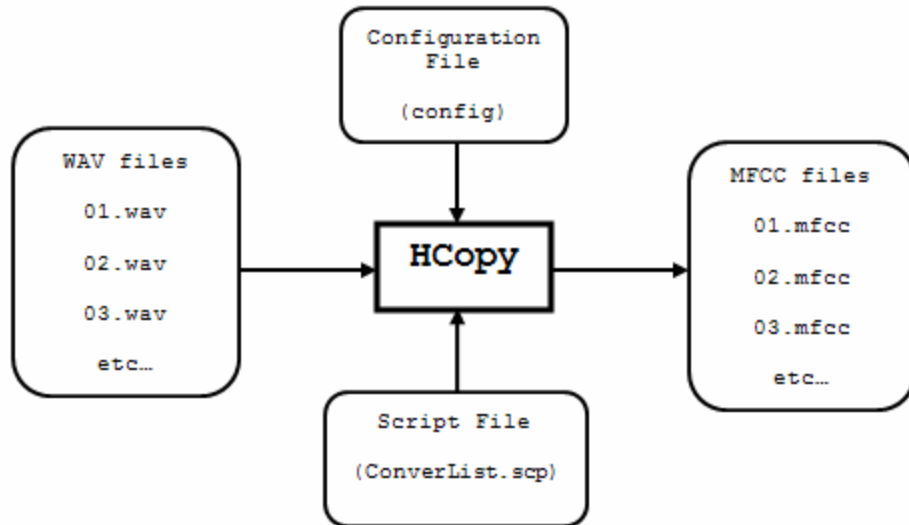
```
# Feature Configuration
TARGETKIND = MFCC_o_D_A
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T

# Source File Format
SOURCEFORM = WAV
SOURCERATE = 454.45
```

The TARGETKIND option specifies the feature representation used. For our wave files an MFCC appended by a Co-cepstral, delta and acceleration coefficients representation was chosen. The coefficients were appended to enhance the performance of the system. TARGETRATE specifies the period between each parameter vector. The successive and overlapping frames are extracted according to the value assigned to WINDOWSIZE. PREEMCOEF sets the pre-emphasis coefficient, this value will compensate for the high-frequency part that was suppressed during the sound production by humans. 0.97 is the typical value used for that coefficient. NUMCHANS, CEPLIFTER and NUMCEPS are the number of filterbank channels, cepstral liftering coefficient and the number of cepstral parameters respectively. The typical values for speech recognizers were set for all three options. SOURCEFORM sets the audio format used. The SOURCERATE was calculated as follows:

$$\text{sample period in nanoseconds} = (1 / \text{audio sample rate}) * 10000000$$

The audio sample rate used for our system was 22 KHz, thus the sample period was set to 454.54 ns.

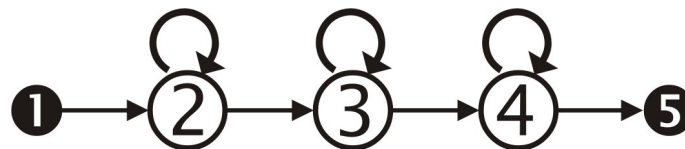


In the command prompt, we ran the HCopy tool with the following parameters:

```
HCopy -A -D -T 1 -C ConfigFiles/config.conf -L Data/Phone_LB -S
ScriptFiles/ConverList.scp > ToolLog/HCopyLog.txt
```

2. Creating an HMM prototype definition

An initial HMM prototype was defined in order to specify the overall characteristics and topology of the model. This prototype is defined by the number of states, dimension and transition to be used. Since we were building a phoneme-based recognizer, each HMM model will represent a single phoneme. Our choice of topology is a three-state topology that is optimal when used with phoneme-based recognizers. This topology consists of non-emitting start and end states and three emitting states. The states are connected in left-to-right way, with no skip transitions.



The HMM Prototype Topology

Here is a detailed explanation of the 3-state HMM topology:

- The first and last small black circles illustrated in the figure represent entry and exit states. These are called null-states or non-emitting states and are only used to concatenate the models.
- The three larger white circles are emitting states.
 1. In the first state, the vocal tract is changing shapes to pronounce the phoneme. This is called the *on-glide* of the phoneme. Some overlap with the preceding phoneme may occur here
 2. The second state is assumed to be *pure* or steady
 3. In the third state, the sound is released and the vocal tract starts a transition to the next phoneme. This is called the *off-glide* of the phoneme. Some overlap with the next phoneme may occur here

These states start from left-to-right and appear in a sequential order with no skip to states that do not follow immediately after.

Each state will be describes as a single-Gaussian observation function, such a function is entirely described by a mean vector and a variance vector. The values assigned to the vectors at this stage of training will be ignored. The actual values will be computed later on. Transition probabilities from one state to another should be given sensible values during the definition but the training process is insensitive to these.

3. Initializing the model with the training data

After creating an initial prototype, we used it to prepare a single HMM with global parameters, which is referred to as a “flat” initialization. “flat initialization” was used because the label files used were not timely aligned.

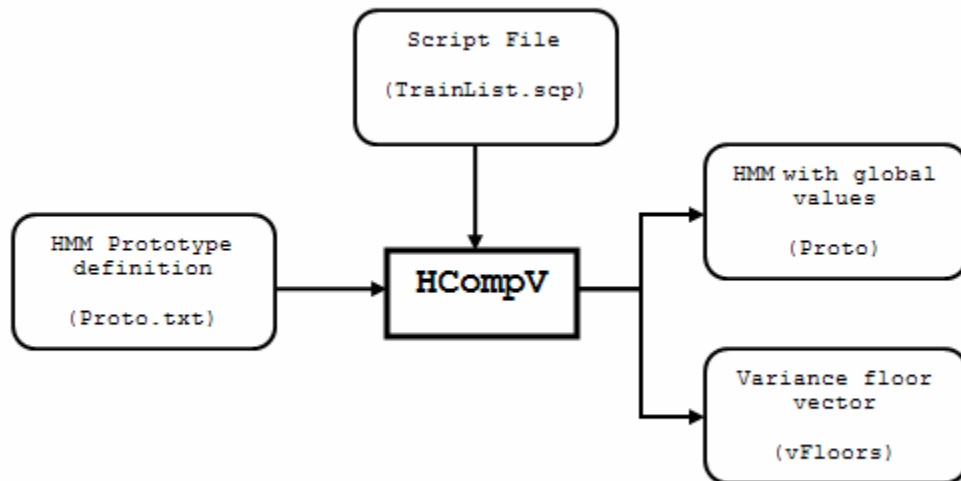
- Why the models are prepared

Two variables are given an initial global value in order to insure fast duplication of the models. These variables are the *mean* and *variance*.
- What HCompV does

HCompV performs a “flat” initialization of the HMM models, where every state in the model is given the same mean and variance vectors.

- HCompV parameters
 1. HMM prototype definition that is stored externally as a text file
 2. List of MFCC files
 3. The configuration file that informs the tool that it will receive MFCC file format
- HCompV output
 - A single HMM with global mean and variance

Using the training data and the HCompv tool, global means and variances were calculated and given to each state in the model. A floor is also set to prevent variances from being badly underestimated through lack of data. Applying HCompv lead to the creation of a HMM with global means and variances, as well as variance vector that sets the floor for the variance values.



In the command prompt, we ran the HCompV tool with the following parameters:

```
HCompV -A -D -T 1 -f 0.01 -L Data/Phone_LB -S
ScriptFiles/TrainList.scp -M Model/hmm0 HMMProto/Proto.txt >
ToolLog/HCompVLog.txt
```

Two files were generated as a result: “Proto” and “vFloors.”

4. Creating the Master Macro File (MMF)

The previous step resulted in two files, one of which defined a single HMM with global means and variances and the other set a floor for the variance vector. Both files needed to be changed in order to be used by the other training tools. A program was created to apply the desired changes to the file. It did the following:

- open “vFloors” and append the following to the beginning of the file:

```
~o <VecSize> 39  
<MFCC_0_D_A>
```

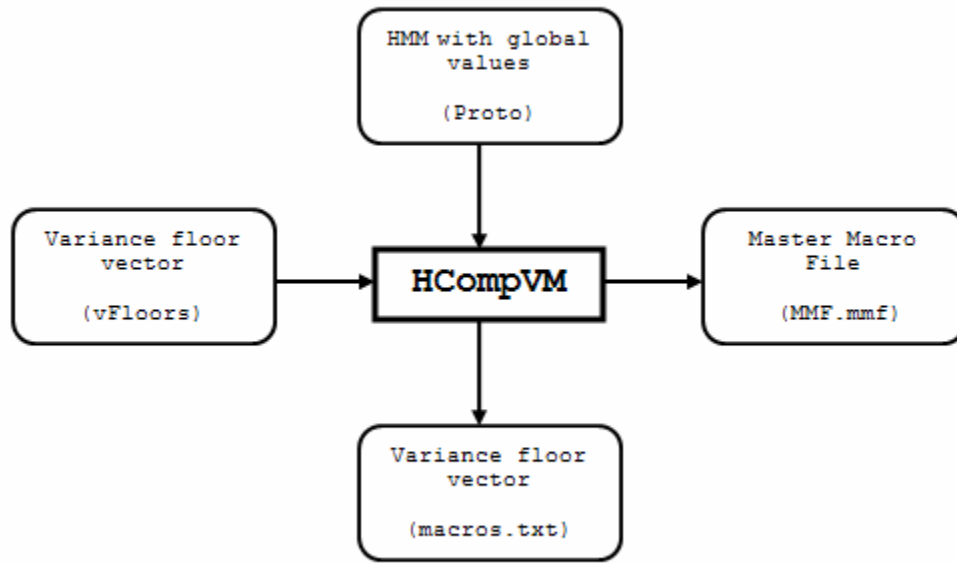
then rename it “macros.txt.”

- Create a Master Macro File (MMF) by opening “Proto” and duplicating the model minus the first three lines for each phone and renaming the model at each time.

The MMF appeared as the following after the change:

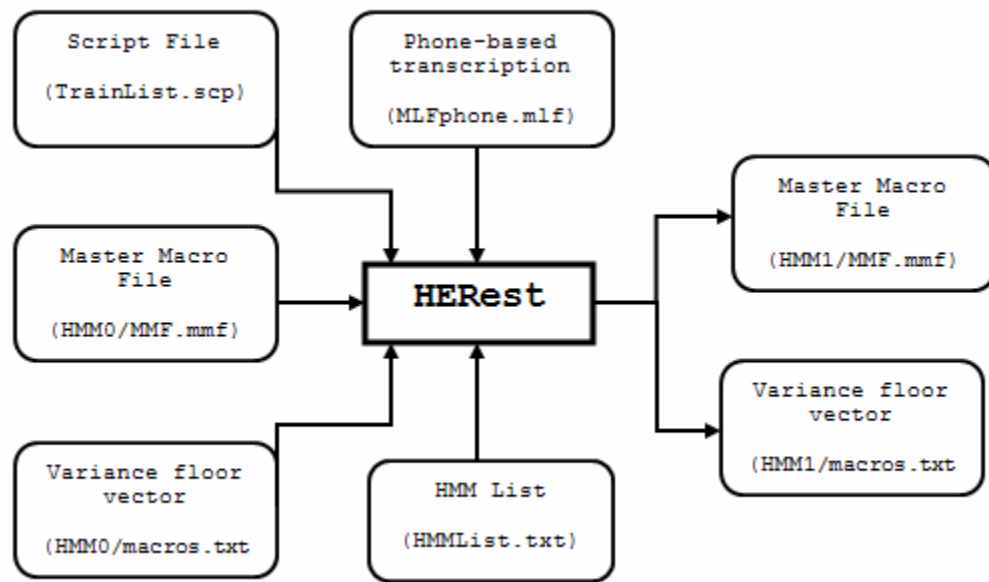
```
~h "a1"  
<BEGINHMM>  
...  
<ENDHMM>  
~h "b1"  
<BEGINHMM>  
...  
<ENDHMM>  
...
```

Most importantly the HMM should be duplicated for each phoneme and be re-named for it. We had a total of 35 phonemes; 34, plus the silence model. The resulting file contained the HMM definitions for all phonemes. The sole purpose of the MMF is to avoid having a large number of individual HMM definition files. The vector size and the feature representation chosen had to be appended to the floor variance vector file.



5. Model Embedded Re-estimation

After duplicating the models and combining them to MMFs, it was time to re-estimate their true values. The flat-start methodology uses the Baum-Welch algorithm to utilize the maximum likelihood criterion and improves on an existing estimate of parameters. HERest is the core HTK training tool. It simultaneously updates all of the HMMs in the system using all of the training data. Each training data is processed in turn and the associated transcriptions are used to construct a composite HMM which spans the whole utterance. The composite HMM is created by concatenating instances of the phoneme HMMs corresponding to each label in the transcriptions. The forward-backward algorithm was then applied and the sums accumulated. After all of the training files have been processed, the new parameter estimates are formed from the sums and new and updated HMMs were created. HERest performs exactly a single iteration. An incremental pruning threshold will be used starting from 250 to 1000, in steps of 2 and 3. The pruning threshold will be used with the backward step of the forward-backward algorithm to reduce computation. The value 1000 of the maximum threshold is recommended in the HTK book. The re-estimation process was performed twice on the MMF.



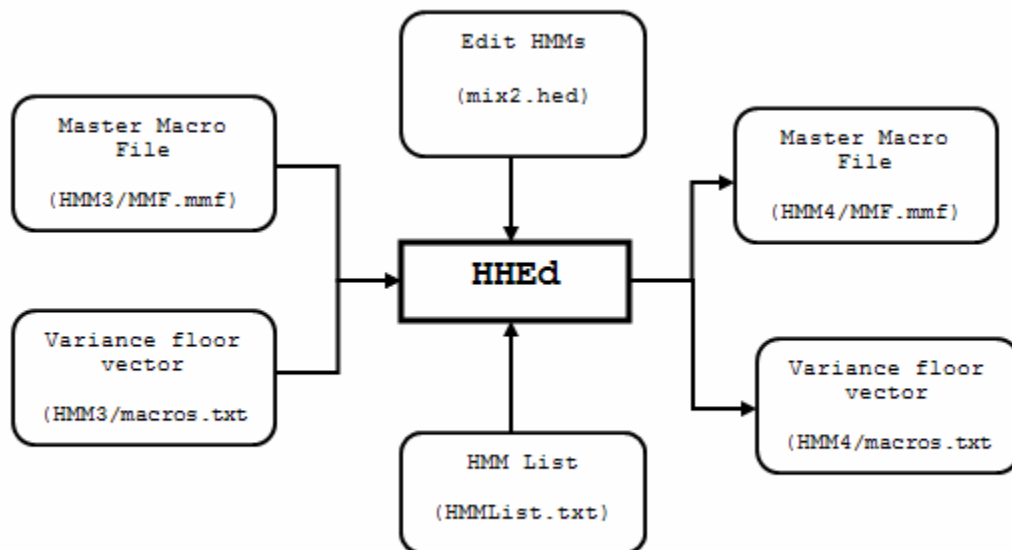
In the command prompt, we ran the HERest tool with the following parameters:

```

HERest -A -D -T 1 -L Data/PhoneSP_LB -I MLF/MLFphoneSP.mlf -t
250.0 150.0 1000.0 -S ScriptFiles/TrainList.scp -H
Model/hmm64/macros.txt -H Model/hmm64/MMF.mmf -M Model/hmm65
Lists/HMMList2.txt > ToolLog/HERestLog42.txt
  
```

6. Increasing Gaussian mixtures

Although this step was optional, applying it tremendously increased the recognition rate. A command used with the HTK tool, HHed, increased the number of components in a mixture by a process called mixture splitting. This approach allowed for better flexibility since it increases the mixtures incrementally until the desired level of performance is achieved. After each mixture increment, two iteration of embedded of training was applied. In all 20 mixtures were created.



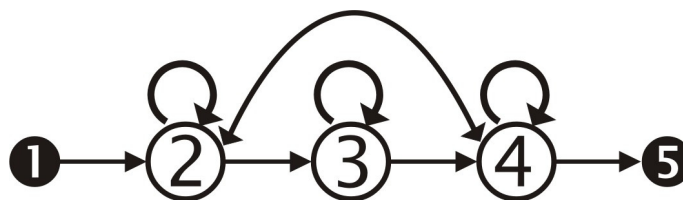
In the command prompt, we ran the HHEd tool with the following parameters:

```

HHEd -A -D -T 1 -H Model/hmm6/macros.txt -H Model/hmm6/MMF.mmf -
M Model/hmm7 CmdFiles/mix2.hed Lists/HMMList.txt >
ToolLog/MixLog2.txt
  
```

7. Fixing the silence model

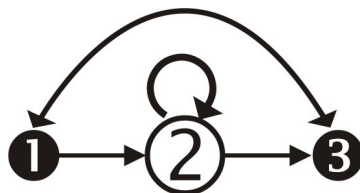
After the models have been trained and started to take shape, it is important to make the models more robust. The silence model was given the same topology as the other phonemes, though it has to take care of periods of silence that may vary drastically in length. Therefore changes must be applied to that model; this was achieved using HHEd which is used to manipulate HMMs. A transition from the 2nd to the 4th state and back from the 4th to the 2nd state was added.



The First Modification Applied to the Silence Model

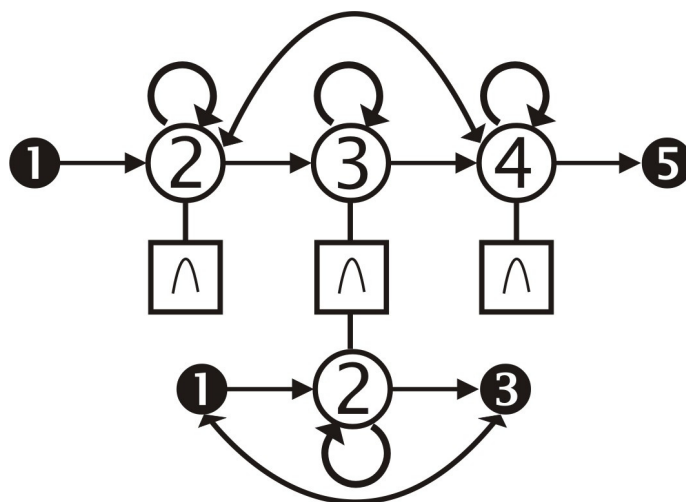
This was done only after the model had received some training, otherwise it would have absorbed a large part of the utterance resulting in badly modeled HMMs.

The previous training steps have been applied without the “short pause” model because of the special nature of the model. The “sp” model is supposed to take care of optional silences between words. Since the “sp” model takes care of very short durations of silence, it was created as a single state HMM which has a direct transition from the entry state to the exit state.



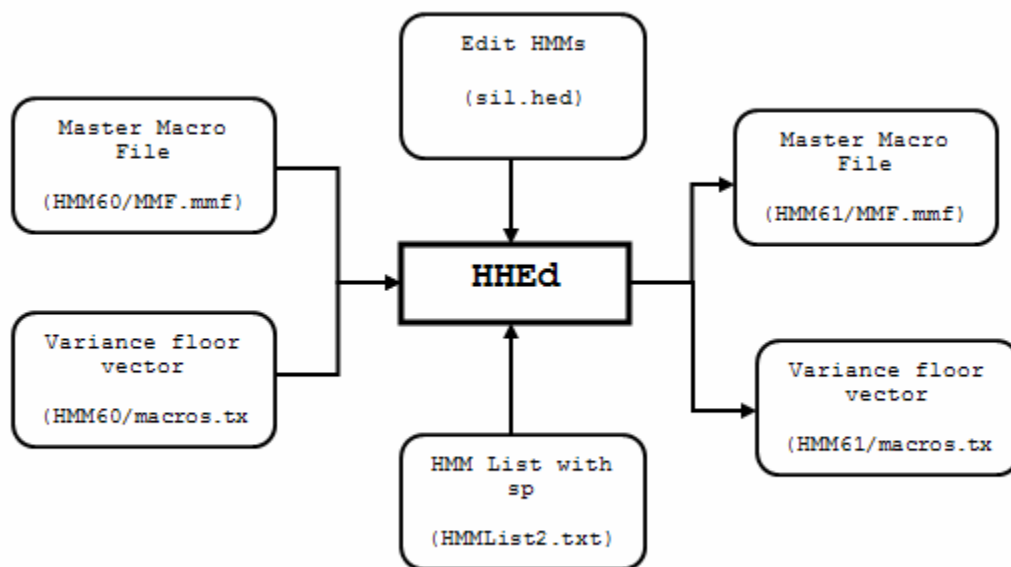
The Short Pause Model “sp”

The single state of “sp” is identical to the middle state of “sil”, therefore a program was created to duplicate the middle state of the “sil” model and append it to the MMF. Then HHed was used in order to tie the lone state in the “sp” model to the middle state of the silence model, now these two states share the same set of parameters.



The Two Models Combined to Create the New Refined Model

Seven iterations of embedded re-estimation were applied using the label files that included the “sp” model.



In the command prompt, we ran HHed with the following parameters:

```
HHed -A -D -T 1 -H Model/hmm63/macros.txt -H Model/hmm63/MMF.mmf
-M Model/hmm64 CmdFiles/sil.hed Lists/HMMList2.txt >
ToolLog/HHedFixSilLog.txt
```

HERest was applied again.

8. Creating Tri-phones

The commands executed so far created what we call a monophone system. A monophone system does not take into account linguistic effects such as coarticulation since it used phoneme that are too fine grained to model these effects. It is assumed in a monophone system that phonemes sound more or less the same in every situation; this is not true in normal speech, since articulation are made quickly and modified by neighboring articulations. To capture this, models will need to take into account the context of the phoneme. To model coarticulation, triphones were used. Triphones takes into account the models context by taking into consideration the left and right neighboring phonemes. Now two exact phonemes that have different neighbors are different triphone models.

The first step to training a triphone-based system was creating triphone transcriptions. This was achieved using HLEd, which created a new triphone list of all possible combination of phonemes found in the monophone-based transcriptions. It also created new triphone label files and a Master Label File (MLF). The “sil” and “sp” model should

remain as monophones by specifying them as word boundary symbols. The triphone label will appear in the following form:

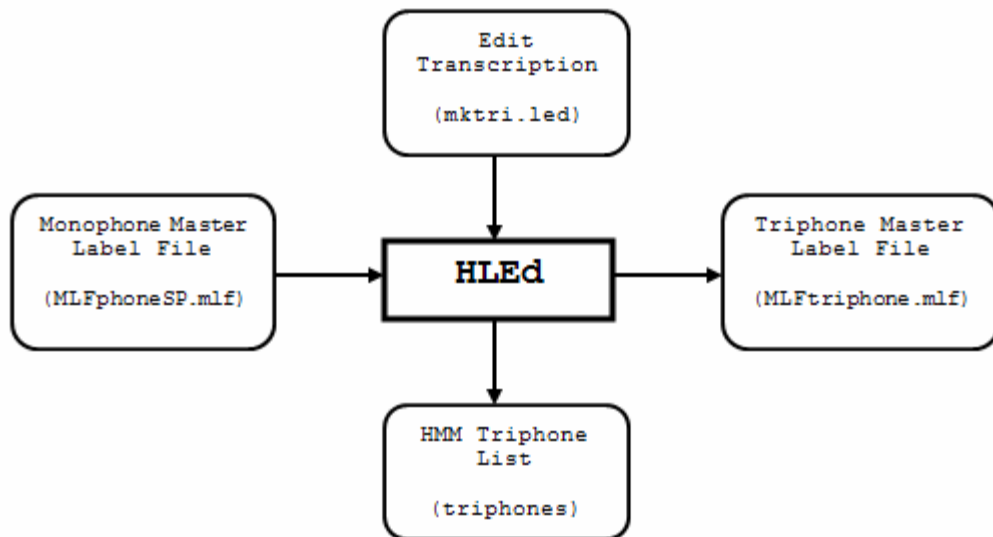
<left context> <phoneme> <right context>

For example the label for the word “sth”, which is the number six in Arabic represented in Roman format, will appear as

sl+tl sl-tl+hl tl-hl

The triphone list created by HLEd was then used by the HMM editor HHed, that cloned each HMM as often as possible and renamed it as a triphone. The number of triphone HMMs created has increased in regard to the previous number of initial models (35 HMMs in all).

Five more iterations of embedded re-estimation were applied using the triphone-based label files.

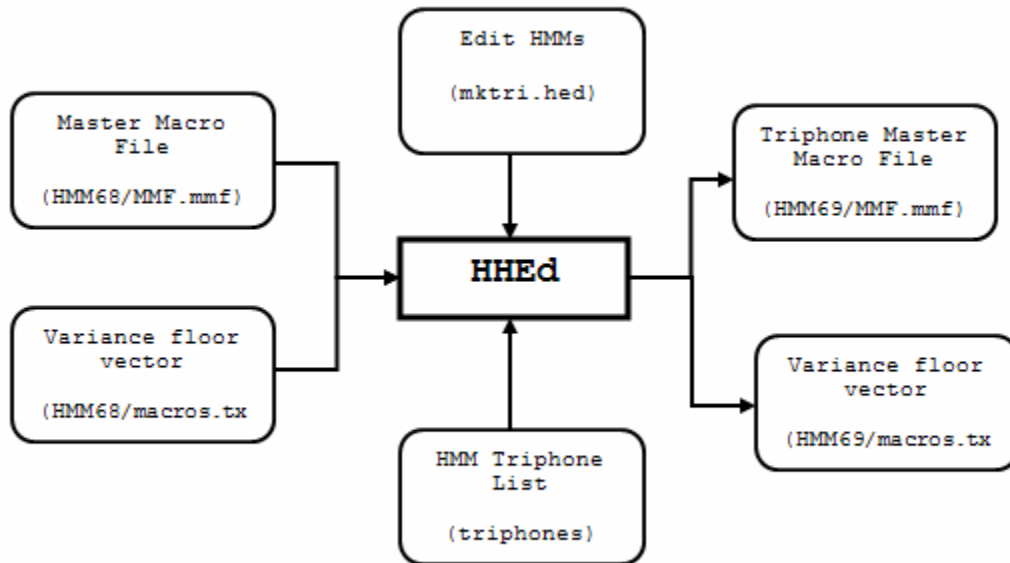


HLEd to create triphone label files:

```
HLEd -A -D -T 1 -n Lists/tri-phones.txt -l Data/PhoneTri_LB  
CmdFiles/tri.hed MLF/MLFphoneSP.mlf > ToolLog/TriLog1.txt
```

HLEd again to create a triphone MLF:

```
HLEd -A -D -T 1 -n Lists/tri-phones.txt -l Data/PhoneTri_LB -i  
MLF/tri-phones.mlf CmdFiles/tri.hed MLF/MLFphoneSP.mlf >  
ToolLog/TriLog2.txt
```



And finally, HHed to create triphone MMF

```
HHed -A -D -T 1 -w Model/hmm67/tri-phones.mmf -H  
Model/hmm67/macros.txt -H Model/hmm67/MMF.mmf  
CmdFiles/mmfttri.hed Lists/HMMList2.txt > ToolLog/TriLog3.txt
```

Stage 3: Specifying a Word Network

Some completely different words sound very similar in their spoken form. This creates a dilemma when dictating to a computer which, unlike human beings, cannot make the appropriate distinction. To solve this, we need to specify a word network that restricts the combination of words. Three approaches exist to specifying a word network: language models, grammar rules, and a simple word loop.

Language Models

Language models are statistical models that are based on estimates of the likelihood and frequency of a word sequence. They are used in continuous speech systems in order to restrict the combination of words. This allows the recognizer make the closest, most accurate guess when different phrases sound similar.

One type of language models is known as bi-grams. Bi-grams estimate the probability of a word sequence using statistical calculations by means of HLStats and HBuild Tools. Language models base their calculations on the context of the corpus, and thus should only be used when the corpus contains sentences that construct a valid meaning. Otherwise we will notice that the accuracy rate will decrease.

Language models are constructed in two steps.

1. Building the Bi-Gram Model Using HLStats:

A bi-gram language model can be built using the HLStats tools. HLStats does the following:

- It takes as input the word-based MLFs
- Builds a table of bi-gram counts in memory and the frequency of the word occurrence
- Outputs a bi-gram with its probabilities

Each bi-gram definition starts with a probability value followed by a sequence of two words. The value specified represents the probability of these two words occurring in the sequence of the text. In the command prompt we executed HLStats with the following parameters:

```
HLStats -A -D -V -T 1 -b LangModels/bigram -S  
ScriptFiles/WordLabelScript.scp Lists/B_wordlist.txt
```

2. Building a Word Network Using HBuild

A word network is a structured way to represents the bi-gram probabilities. It is the format that recognition tools can identify with and accept. HBuild does the following:

- It takes as input the bi-grams
- Outputs a word network representing those bi-grams

The output word network is stored using the HTK Standard Lattice Format (SLF) that uses a list of nodes to represent words and a list of arcs to represent transitions between words. The transitions have the appropriate probabilities attached to them. In the command prompt we executed HBuild with the following parameters:

```
HBuild -A -D -V -T 1 -m LangModels/bigram Lists/B_wordlist.txt  
LangModels/BigramNET.slf
```

Grammar Rules

Grammar rules are based on the Backus Naur Form (BNF) ([Appendix B – Other Tools and resources](#)) which allows us to selectively specify the allowable sequence of words. BNFs are created manually, and are suitable in cases where the corpus does not contain meaningful sentences or is relatively small.

Word Loop

Word loops allow any sequence of words that exist in the dictionary. They are implemented when any sequence of words is allowed in the application.

Stage 4: Formal Testing of System Accuracy

A formal method exists to test ASR systems for accuracy. Using the HResults tool, the training wave files are submitted and accuracy rates are calculated on those wave files. The resulting rate is a formal description of the maximum accuracy rate the system can achieve. When recognition is performed on real data (new data acquired from the user of the final system) it is expected to generally be lower.

When HResults is used to calculate the sentence accuracy, the basic output is recognition statistics for the whole file set in the following format:

```
----- Overall Results -----  
SENT: %Correct=13.00 [H=13, S=87, N=100]  
WORD: %Corr=53.36, Acc=44.90 [H=460, D=49, S=353, I=73, N=862]
```

The first line gives the sentence-level accuracy based on the total number of label files which are identical to the transcription files. The second line is the word accuracy based on matches between the label files and the transcriptions. In this second line, H is the number of correct labels, D is the number of deletions, S is the number of substitutions, I is the number of insertions and N is the total number of labels in the defining transcription files. The percentage number of labels correctly recognized is given by

$$\%Correct = H/N * 100\%$$

and the accuracy is computed by

$$Accuracy = H-I/N * 100\%$$

Acc describes the estimation of maximum accuracy rate the system can reach, and thus is the rate that concerns us. In the appendix ([Appendix C – Our Steps](#)) you will find the

results for each of the systems we implemented using the different strategies, including the final one embedded into AraDict.

8.1.2 Strategy1: Towards a Speaker Independent System

Requirements for this strategy

Several requirements exist to develop a speaker independent system. First, we needed a large speech corpus, containing multiple speakers with distinct acoustic characteristics. After submitting it to the generic process, an optional adaptation session for each user can sometimes increase the accuracy rate.

The Speech Corpus

Description of SAAVB Speech Corpus

After requesting a speech corpus from King Abdul-Aziz City of Science and Technology (KACST) ([Appendix D – External Communications Log](#)), we were granted a small sample of the Saudi Accented Arabic Voice Bank (SAAVB) speech corpus. From here onwards we will refer to this sample as the voice bank.

The voice bank contained 15 speakers in total; 10 male and 5 female, each having distinct acoustic characteristics. Data was divided according to speaker, each in a separate folder. Each folder contained the following:

1. A single text file containing information about the speaker (*.txt)
2. 60 recorded wave files (*.wav)
3. The Arabic transcriptions of wave files, with diacritics (*.txt)
4. The Arabic transcriptions of wave files, without diacritics (*.txt)

The same 60 sentences were repeated by all 15 speakers resulting in the lack of variation in content. The subject matter of these sentences could not be classified as a particular field. The sentences contained numbers, dates, commands, names of some local cities and villages, names of people, names of companies, some verbs and nouns. Most sentences did not construct a valid meaning.

The Wave Files

The wave files have been recorded by the speakers themselves using mobile phone devices. In total, the sample contained 900 recorded files. Many, however, contained highly noticeable environmental disturbances such as background noise, as well as disfluencies in speech such as truncated words and slips of tongue. Since the result of

training highly depends on the quality of the recorded files, the poor files had to be manually removed from the voice bank. After selectively removing the poor files we ended up with approximately 750 wave files.

The Transcriptions

Two kinds of transcriptions existed for each wave file. The difference was:

- a. Numbers were spelled in the first kind (واحد، اثنان، ثلاثة... etc.) and enumerated in the second (1...2...3...etc.)
- b. Text was partially diacritic in the first kind and non-diacritic in the second.

Rather than spelling the words in standard Arabic, the transcriptions were written to match the pronunciation of the particular speaker. For example, instead of writing (خمسة عشر) the transcriptions spelled (طعش خمس). This was manually adjusted for each individual file. Diacritics were added to the first kind to make it fully diacritic rather than partially.

Our Approach

The amount of adjustment made to the voice bank poses a very valid question: why was it adjusted instead of using a different corpus? Why didn't we record our voices instead? SAAVB was perhaps constructed for a slightly different purpose; a telephone-based recognition system, but the scarcity of Arabic training corpuses limited our choices. The second option of recording our own corpus was not suitable for this strategy. As we mentioned, for a speaker independent system we needed speakers with distinct acoustic characteristics. These characteristics did not necessarily apply to us. This method however was adapted in the strategies that followed.

The Prototypes

The structure of the transcriptions led us to construct three different prototypes to train. Prototype 1 utilized the first kind of transcriptions that we submitted to modifications, making it fully diacritic and spelled in standard Arabic. Prototype 2 also utilized the first kind of transcriptions but with no modifications, making it partially diacritic with spelling matching the speaker's pronunciation. Prototype 3 utilized the second kind of transcriptions with no modifications, making it non-diacritic with spelling matching the speaker's pronunciation.

	Transcriptions	Diacritics	Spelling
Prototype 1	First kind	Fully	Corrected to standard Arabic
Prototype 2	Fist kind	Partially	Matching the speaker's pronunciation
Prototype 3	Second kind	Non-diacritic	Matching the speaker's pronunciation

The goal behind constructing three prototypes was to measure the accuracy rate for each, then select the best to embed into the final system.

Speaker Adaptation

Effective speaker adaptation enables a speaker independent system to adapt to the characteristics of a new speaker given very small amounts of training data. These amounts of training data are well below those required for training speaker dependent systems. The user records the data then uses the speaker adaptation tools provided with HTK to process this data.

To perform speaker adaptation we did the following:

- Selected a small sample of training utterances
- Transformed to feature data
- Selected speaker and environment for adaptation
- Created word-level transcriptions
- Generated the adaptation data

First Step

```
HHed -B -H Model/hmm73/macros.txt -H Model/hmm73/MMF.mmf -M classes
adapt/regtree.hed Lists/triphones > ToolLog/HEEd.txt
```

-M classes: Directory to store the regression class tree in. regtree.hed contained the following commands:

```
LS "hmm15/stats
RC 32 "rtree"
```

The LS command loads the state occupation statistics file stats generated by the last application of HERest which created the models in hmm15.

The RC command then attempts to build a regression class tree with 32 terminal or leaf nodes using these statistics.

Second Step

```
HERest -C ConfigFiles/config -C adapt/config.global -S adapt/adapt.scp -I  
adapt/adaptPhones.mlf -H Model/hmm73/macros.txt -u a -H  
Model/hmm73/MMF.mmf -z -K adapt mllr1 -J classes Lists/triphones >  
ToolsLog/HERestAdapt1.txt
```

Adapt/config.global is a configuration file contains

```
HADAPT:TRANSKIND = MLLRMEAN  
HADAPT:USEBIAS = TRUE  
HADAPT:BASECLASS = global.txt  
HADAPT:ADAPTKIND = BASE  
HADAPT:KEEPXFORMDISTINCT = FALSE  
HADAPT:TRACE = 61  
HMODEL:TRACE = 512  
HADAPT:TRANSKIND = MLLRMEAN
```

Maximum likelihood linear regression (MLLR) is the kind of adaptation transforms we used. MLLR computes a set of transformations that reduce the mismatch between an initial model set and the adaptation data. More specifically MLLR is a model adaptation technique that estimates a set of linear transformations for the mean and variance parameters of a Gaussian mixture HMM system.

HADAPT:BASECLASS = global.txt

specifies the set of the components that share the same transform. We used a global transformation for all components. The global file contains:

```
~b ``global"  
<MMFIDMASK> *  
<PARAMETERS> MIXBASE  
<NUMCLASSES> 1  
<CLASS> 1 {*.state[2-4].mix[1-20]}
```

Based on:

```
HADAPT:ADAPTKIND = BASE
```

We used:

```
HADAPT:TRACE = 61
```

```
HMODEL:TRACE = 512
```

Parameters:

-u a : updates the HMM linear transform.

-z tmf: Save all output transforms to filetmf

Third Step

```
HERest -a -C ConfigFiles/config -C adapt/config.rc -S adapt/adapt.scp -I  
adapt/adaptPhones.mlf -H hmm71/macros.txt -u a -H hmm71/MMF.mmf -J adapt  
mllr1 -K adapt mllr2 -J classes Lists/triphones > ToolLog/HERestAdapt2.txt
```

The second time we use the HERest command we use the regression tree Config.rc which contains:

```
HADAPT:TRANSKIND = MLLRMEAN  
HADAPT:USEBIAS = TRUE  
HADAPT:REGTREE = rtree.tree  
HADAPT:ADAPTKIND = TREE  
HADAPT:SPLITTHRESH = 1000.0  
HADAPT:KEEPXFORMDISTINCT = FALSE  
HADAPT:TRACE = 61  
HMODEL:TRACE = 512
```

Minimum occupancy to generate a transform

```
HADAPT:SPLITTHRESH = 1000.0
```

Save transforms in a transform model file (TMF)

```
HADAPT:KEEPXFORMDISTINCT = FALSE
```

Macroname of regression tree

HADAPT:REGTREE = rtree.tree

Parameters:

- J adapt mllr1: mllr1 is the input transforms found in adapt directory.
- K adapt mllr2: Output transform directory and optional extension for output transforms. The default option is that there is no output extension and the current transform directories used.

Fourth Step

```
HVite -H Model/hmm74/macros.txt -H Model/hmm74/MMF.mmf -S testing.scp -J  
adapt mllr2 -k -i recoutAdapt.mlf -w LangModels/loop.slf -J classes -C  
ConfigFiles/config -p 0.0 -s 5.0 Dictionary/dictionary.txt Lists/triphones  
>ToolLog/HViteAdapt.txt
```

Recognize files in testing script using the final transforms in adapt directory.

Results and problems

Accuracy rates

After completing all the steps, the three prototypes were ready to be tested for accuracy using the formal method described in the generic process. For detailed results see [Appendix C – Our Steps](#). The following table compares the accuracy rates of the three prototypes, using different combinations of training options and word networks.

	Mono-phones + Word Loop	Mono-phones + Bi-grams	Tri-phones + Word Loop	Tri-phones + Bi-grams
Prototype 1	Acc=40.66 %	Acc=28.94 %	Acc=59.84 %	Acc=46.50 %
Prototype 2	Acc=33.69 %	Acc=54.93 %	Acc=75.87 %	Acc=76.02 %
Prototype 3	Acc=36.65 %	Acc=20.21 %	Acc=57.47 %	Acc=26.70 %

We notice that prototype 2 produced the best results, which contained transcriptions that matched the speaker's pronunciation and was partially diacritic. We also conclude from the table that applying tri-phones and a simple word loop produced the best overall results. Applying language models in most cases decreased the recognition. Prototype 2 produced the highest results and thus was chosen among the three prototypes.

Testing the systems on our voices

The results indicated that the system was properly trained and was ready to be tested with our voices. At this stage we faced some difficulties. When we tested our voices, the system remained in a waiting state as if it was not receiving input from the microphone. After investigating the matter and seeking technical help ([Appendix D – External Communications Log](#)) we were introduced to the issue of sampling rate.

Sampling Rate

The sampling rate, measured in hertz (Hz) defines the number of samples per second taken from a continuous signal to make a discrete signal. Since SAAVB was developed for a telephone-based application, the sampling rates of the digitally recorded files were modified to suit this application. Analog phones typically sample data at a rate of 8,000 Hz while modern sound cards in PC's sample data at a rate of 22,000 Hz or 44,000 Hz. There was a mismatch in the sampling rates of training data and data to be recognized which causes the frequencies to shift and acoustic models did not match. This explained why the system was held at a waiting state, not detecting data with higher frequencies.

Two solutions were proposed to us from technical resources ([Appendix D – External Communications Log](#)). The first was to adjust the settings in the configuration file, so that the system would expect a different sampling rate for input. A second solution was to record our voices every time we wanted to input data at sampling rate of 8,000 Hz, using a special software, in order to match the training data. Both ways were tested none came out with satisfactory results.

Adjusting the Configuration Settings

The SOURCERATE in the configuration file is described as “sample period of source in 100 ns units” so we can say [sample period = 1/ sampling rate] then we multiply by 10000000 in order to convert to nanoseconds. Initially the source rate was 1250 based on:

$$(1/8000)*10000000 = 1250$$

We changed the source rate in the configuration file to 454.54 based on the equation:

$$(1/22000)*10000000 = 454.54$$

This was the first proposed solution. The system however was still stuck in a waiting state expecting input.

Recording the New Data Using Similar Sampling Rate

Using a software called GoldWave, we recorded the new data with a sampling rate that matched the voice bank's. This was the second proposed solution. It was tested with all three prototypes; all yielded extremely poor results.

Alternatives

At this stage we needed to consider alternatives. To make sure that the problem was a matter of incompatibility between the training data and the new data, we considered replacing only the wave files in the voice bank then reprocessing it in the same manner.

8.1.3 Strategy2: Replacing the Wave Files

Our Approach

We divided the 15 speakers among the 7 of us and each of us recorded some part of the voice bank. Although the same amount of data as the previous strategy was processed, we cannot claim that this is a speaker independent system for two reasons: first, the number of actual speakers was decreased by almost half, and second, the speakers needed to have distinct acoustic characteristics, which did not necessarily apply to us. We cannot call it a speaker dependent system either because the recordings belonged to several speakers rather than a single speaker.

Results and problems

After going through the whole process, including speaker adaptation, we tested the system first using the HResults tool. The accuracy rates were relatively good. For detailed results see [Appendix C – Our Steps](#).

	Mono-phones + Word Loop	Mono-phones + Bi-grams	Tri-phones + Word Loop	Tri-phones + Bi-grams
Recorded System	Acc=55.36 %	Acc=64.96 %	Acc=72.96 %	Acc=74.56 %

However, once we proceeded to test the system with our voices, the recognition was extremely poor. But, unlike before, it was not stuck in a waiting state and was able to detect our input, which confirms a form of incompatibility. But since most of the words were incorrectly recognized we had to consider a different approach to our system. As a last resort, we implemented a speaker dependent system.

8.1.4 Strategy2: Towards a Speaker Dependent System

Requirements for this Strategy

This strategy requires each user to record his/her voice individually. Each training session will contain only recordings of a single speaker. Since making the system speaker dependent limits its usability, we decided to equip it with the facility of adding a new user dynamically to extend its usability.

Our Approach

Re-Creating the Speech Corpus

All of our previous attempts involved using the voice bank resulting in poor recognition due to the unsuitability of the voice bank to the nature of our project. We decided to create our own training data for this strategy instead. We first started with a simple number system, and slowly increased it to include more words such as courses, then names of people, testing it each time on real data to ensure good recognition results. We ended up creating a system with a vocabulary size of approximately 50 words. For the complete list of words see [Appendix C – Our Steps](#).

Transcriptions

We wrote transcriptions of continuous sentences including random combinations of the words we implemented, then processed the system and tested it. Whenever the recognizer confused two or more words we added more transcriptions with a different mixture of these words in order to train it to better distinguish them and tell them apart. We repeated this process several times until the results were satisfactory. In total we ended up with 66 transcription files.

Wave Files

We selected three team members at random to read the transcriptions and record their speech. Of course each recorded her files individually and trained independently of the others. All three resulted in almost identical accuracy rates.

Word Network

First we implemented it using BNFs. But when we decided to implement the cell skipping command, BNFs were no longer suitable for this task. So instead we implemented a simple word loop.

Developing Dynamic Training Program

Since the application will be speaker dependent, it was a good idea to implement the functionality of adding a new user. We already had the programs we created for the generic process, and we were able to combine these into a single program that does the training dynamically.

Results

When tested using the formal method, this strategy produced the highest accuracy rate achieved so far since the beginning of the project: 89.2% ([for detailed results see Appendix C – Our Steps](#)). The recognizer was now ready to be integrated with the add-in application.

8.2 Running the Recognizer

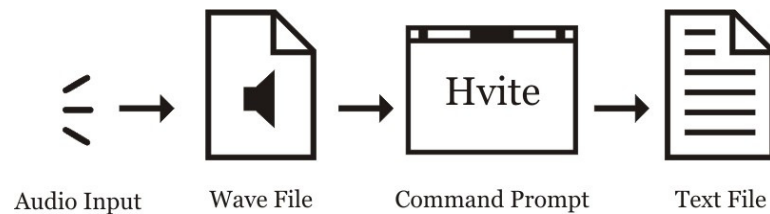
After the recognizer was built and tested, it was time to select a method for operating it. Two ways exist to run the recognizer: offline and live. HTK toolkit provides the recognition tool HVite, which uses the token passing algorithm. It is used for both offline as well as live recognition. The difference between the two exists in the configuration file.

8.2.1 Offline Recognition

Offline recognition requires us to first record the speech input as a wave file, then execute a recognition command that takes that wave file, processes it, and then produces results.

How it Works

Speech is recorded into a wave file using sound recording software. Then, in the command prompt the HVite command is executed on that wave file and an output text file name is specified. After executing the command, the specified text file will contain the textual recognition results.



Offline Recognition

Configuration and Command

It uses the same configuration settings used in the training process which is the following:

```
# Feature Configuration
TARGETKIND = MFCC_0_D_A
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T
# Source File Format
SOURCEFORM = WAV
SOURCERATE = 454.45
```

In the command prompt, we execute Hvite with the following parameters:

```
HVite -C ConfigFiles/config -H Model/hmm74/macros.txt -H  
Model/hmm74/MMF.mmf -w LangModels/loop.slf -p 0.0 -s 5.0 -o TS  
Dictionary/dictionary.txt Lists/triphones % 1
```

HVite will take as input the allowable word sequence, in our case a simple word loop was used, the Master Macro File (MMF) accompanied by the variance vector floor file, dictionary, triphone HMM list and the recorded file saved as (1.wav).

Our Approach

The idea was to hide the entire execution process from the user of the application by implementing offline recognition as a black box, and thus the only parts visible to the user are acquiring data and outputting the result as text directly into the application.



Offline Recognition as a Black Box

This was done by means of a program that automated the process ([Appendix C – Our Steps](#)). The program invoked sound recording software Total Recorder ([Appendix B – Other Tools and Recourses](#)), saved the data in an output file, invoked HVite and printed the contents of the output file on the screen in the application.

An Issue

The main issue concerning offline recognition is that the processes of acquiring data and recognizing data cannot occur in parallel; instead we have to constantly alternate between the two, which meant that the user only observes his spoken words when he completely finishes speaking and submits his speech. On the other hand, offline recognition tends to produce the best results when it comes to accuracy and reliability.

Another issue is that the single recording can be no longer than a minute. This is because the Total Recorder version used is a limited test version. The complete developer version needs to be purchased for the feature of extending the recording time. Because Total

Recorder was only embedded to AraDict towards the very end, there was no time to purchase and test the complete developer version.

Delay

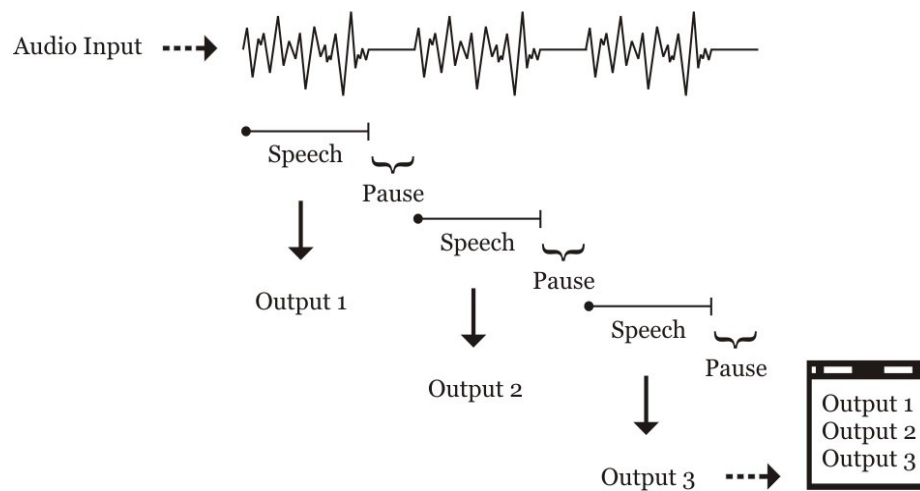
Recorder recognition produces instant results. The only delay is caused by switching between the add-in and the Total Recorder.

8.2.2 Live Recognition

An alternative to offline recognition is live recognition which accepts live input directly from the user without the need to record into a wave file prior to recognition. Live recognition is not necessarily instant recognition; it simply eliminates some of the steps in between.

How it Works

Live recognition is run directly from the command prompt. HVite is executed after specifying that the input is live, which allows it to acquire the speech input directly from the connected microphone. It continuously accepts input, and whenever a pause in the speech is detected it processes the speech prior to the pause assuming it is a whole sentence while continuing to acquire more input from the microphone.



Live Recognition

Configuration and Command

It uses the following configuration:

```
# Feature Configuration
TARGETKIND = MFCC_0_D_A
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T

# Source File Format
SOURCEFORM = HAUDIO
SOURCERATE = 454.45
```

The source file format was changed to HAUDIO in order to accept input from the microphone. It is essential to set the ENORMALISE option to true during training to be able to recognize live speech.

In the command prompt, we execute Hvite with the following parameters:

```
HVite -C ConfigFiles/live.cfg -e -H Model/hmm74/macros.txt -H
Model/hmm74/MMF.mmf -w LangModels/loop.slf -p 0.0 -s 5.0 -o TS
Dictionary/dictionary.txt Lists/triphones
```

The only difference to the previous commands is that no input wave file is specified at the end.

Our Approach

Live recognition was also implemented as a black box. A function ([Appendix C – Our Steps](#)) invoked HVite as a hidden background application and outputted the results into the application.



Live Recognition as a Black Box

An Issue

Although acquiring data and recognizing data occur in parallel, the process is somewhat slow due to the processing that occurs and constant detection of pauses in speech. In contrast to offline recognition it is noticeably much slower.

It also produces results with lower accuracy. In offline recognition, the start and end of sentences are easily identified, but in live recognition, sometimes brief pause between words is mistaken for the end of sentence silence, which results in dropping the words following that pause, which in turn causes lower accuracy rates and is thus unreliable.

Real Time Recognition

Real time recognition is a form of live recognition that produces “real time” or instant results without any delay. There is no facility in HTK that enables real-time recognition. And thus we needed to search for an external facility. An application toolkit for HTK (ATK) was found ([Appendix B – Other Tools and resources](#)). When we tested it using a sample provided with it, it produced bad results. We were also unable to integrate it into the add-in application. After a few weeks of seeking technical help and trying to solve these two issues we decided not to use ATK.

8.3 Developing the MS Add-In Application

8.3.1 MS Word 2007

Ribbons

MS Office 2007 contains a number of new features, the most notable of which is the new graphical user interface called the Ribbon, replacing traditional menus and toolbars. The ribbon is a pane that contains controls, such as buttons and icons, which are organized into a set of tabs, each one containing a grouping of relevant commands. The ribbon is designed to make the features of the application more discoverable and accessible with fewer mouse clicks as compared to the menu-based UI.

Xml

Microsoft Office 2007 bases its entire Graphical interface design on XML, and thus AraDict's interface was implemented using XML as well.

8.3.2 Development and Integration

The add-in application was developed using Visual Studio .NET; programs were implemented using C# and the interface was designed using XML. HVite recognition tools could not be called directly from a C# program so instead we implemented it using callbacks to a batch file that contained the command.

Creating the ribbon

The ribbon was created by means of a code called RibbonX which follows an XML Schema provided by Microsoft. This XML code specifies everything about the user interface. Any additional UI elements (buttons, bars, menus...etc.) we needed to add were inserted into this code. When the code is built, the ribbon appears in MS Word 2007 the next time it is opened.

Build an Office COM Add-In

Microsoft Office XP, Microsoft Office 2003, and Microsoft Office 2007 support uniform design architecture for building application add-ins to enhance and to control Office applications. This design architecture is called Microsoft Component Object Model (COM).

COM Add-In provides five built-in functions: OnConnection, OnDisconnection, OnAddInsUpdate, OnStartupComplete, and OnBeginShutdown. Our entire code was

inserted into OnConnection, the rest were left empty. For implemented functions see [Appendix C – Our Steps](#).

Problems Encountered During Integration

1. Hvite was supposed to terminate whenever the “Stop” button was pressed. There are no means to terminate Hvite except by closing the command prompt while it was running. Since the command prompt needed to be hidden from the user we needed to run a Kill command in DOS. This was also done by means of a batch file called by a C# function.
2. Hvite normally saves the output in a single file. The file could not be used by two different processes at the same time (Hvite and our application). To solve this problem, an option was used in the Hvite command that allows decomposing the output of HVite into a number of files, each of which could be taken by the application and processed thus working in parallel with HVite.
3. We were not able to run the executable files directly using C#. To solve this problem, we ran them using MSDOS Batch Language which saves commands into Batch files with .bat extension and we were able to run them directly in the same way any executable file could be run.

9 Testing

Test Cases	Options	Expected Result	Actual Result	Observations
Testing recorded recognition using command prompt		89.2% of the words should be correctly recognized	All the recorded files were correctly recognized	The recognition results are highly dependent on the environment
Testing live recognition using command prompt		We cannot expect an accuracy rate from the live recognition because it cannot be calculated using a formal method	Some words were correctly recognized, others were incorrectly recognized	The live recognition is highly unpredictable and much less reliable than recorded recognition
Testing recorded recognition on AraDict (GUI)	The creation of an output text file visible only to the developer but not to the user	The output file should be created containing Arabic characters and digits (if any)	Same as expected	
	The time needed for recognized text to be typed on the document after clicking "Display"	When "Display" is clicked, the output should be typed	Some delay occurred (2-4 seconds approximately)	Clicking the display again when a delay occurs speeds up typing the words
	The deletion of some hidden files, generated during the recognition, after processing them	The files should be deleted from the user's folder	Same as expected	
	The correctness of the displayed output, when user uses his correct account	89.2% of the words should be correctly recognized	Most of the words were correctly recognized	The recognition results are highly dependent on the environment

Test Cases	Options	Expected Result	Actual Result	Observations
	The correctness of the displayed output, when user uses someone else's account	Since AraDict is speaker dependent, it is not expected to correctly recognise input from a speaker using someone else's account	Most of the words were incorrectly recognized	If someone uses someone else's account the resulting recognition is most likely poor
	Clicking "Display" without recording anything	The system should type nothing	Same as expected	
	Clicking "Display" more than once	The system should type nothing	The last typed file will be typed again	
Testing live recognition on AraDict (GUI)	The creation of an output text file visible only to the developer but not to the user	The output file should be created containing Arabic characters and digits (if any)	Same as expected	
	The time need to display the output on the document	As the user speaks, the output should be typed on the document	The system does not type anything until the user clicks "Stop"	Although acquiring speech and recognizing it occur in parallel, the we were unable to concurrently display the results as the user spoke
	The deletion of some hidden files, generated during the recognition, after processing them	The files should be deleted from the user's folder	Same as expected	

Test Cases	Options	Expected Result	Actual Result	Observations
	The correctness of the displayed output, when user uses his correct account	We cannot expect an accuracy rate from the live recognition because it cannot be calculated using a formal method	Some words were correctly recognized, others were incorrectly recognized	The live recognition is highly unpredictable and is much less reliable than recorded recognition
	The correctness of the displayed output, when user uses someone else's account	We cannot expect an accuracy rate from the live recognition because it cannot be calculated using a formal method	None of the words were correctly recognized	The live recognition is highly unpredictable and is much less reliable than recorded recognition
	Clicking "Stop" without saying anything	The system should type nothing	Same as expected	
Testing "Add a New User"	Entering the same user name	The system should refuse repeated usernames	The system will overwrite an old user with the same name	We did not consider this case in our program
	Re-recording a transcription file	The system should overwrite the previously recorded file	Same as expected	
	The cancellation of the process	The system should cancel the process and delete the user's folder and all its contents	Same as expected	If the folder or one of the files was in use, the folder or that file will not be deleted
	Required time for dynamic training	Should not exceed 5-6 minutes	Same as expected	
	Progress bar	Should terminate at the end of the training process	It continues running after the end of the training process for a few minutes then terminates	

Test Cases	Options	Expected Result	Actual Result	Observations
	The adding of the username to the list of existing users in the drop down menu when training is finished	The user name should be listed in the “Existing Users” list	Same as expected	If the user name is not listed, the user should close the Office Word window and reopen it
Testing the system using a blank document rather than templates	Where in the document the output will be typed	The output should be typed at the position of the cursor	Same as expected	The user needs to manually changes of the place of the cursor; there is no speech command to do this
	The attempt to move the cursor's position using the word انتقل	The word انتقل will be typed on the document instead of moving the cursor	Same as expected	When a blank document is used in dictation, cell-skipping commands are disabled
Testing the system using any of the three provided templates	The attempt to move the cursor's position using the word انتقل	The cursor moves to the next cell in sequence	Same as expected	The action will take place when the word انتقل is correctly recognized and not mistaken for another word. Like any other word, the WER is 10.8%
	The position where the output will be typed if the user changes the place of the cursor manually	The output will be typed at the new position the user specified	The system will ignore the user's action and type the output at the position following the same sequence	The user must skip each cell using the command انتقل until the desired cell is reached
Testing the system when more than one template is open	The template in which the output will be displayed	The output should be typed on the template where the user clicks “Display” or “Stop”	Same as expected	

10 Future Work

1. Improve Live Recognition

Live recognition was a very difficult task to implement. We will continue to experiment with it and try to improve it by further research and experimentation using ATK.

2. Improve Recorded Recognition

Since the complete developer version will be purchased, we will be able to extend the time of recordings and hide the recorder from the user.

3. Develop Generic Templates

To make our system more useable we will create general template for non-specific usage such as letters, reports, tables...etc.

4. Facilitate Adding New Words

To extend the usage of our system we will enable the user to add new words of his choice and enable the system to insert these words into the training corpus, retrain the system the system with the new words and insert the words into the dictionary. This is however will be a lengthy process.

5. Develop as Add-In to MS Excel 2007

The add-in application we developed was specific to word. Several changes can be made to the application to make it an add-in to MS Excel.

11 Bibliography

- [1] Jackson M. Automatic Speech Recognition: Human Computer Interface for Kinyarwanda Language. Makerere University, 2005.
- [2] Speech Recognition, Wikipedia, From, http://en.wikipedia.org/wiki/Speech_recognition
- [3] Technology Overview, Speech Recognition, Microsoft, Microsoft Speech Server, from, <http://www.microsoft.com/speech/evaluation/techover/>
- [4] What is speech recognition, comp.speech Frequently Asked Questions, Carnegie Mellon University, from, <http://www.speech.cs.cmu.edu/comp.speech/Section6/Q6.1.html>
- [5] Latency, Wikipedia, from, <http://en.wikipedia.org/wiki/Latency>
- [6] Connected Speech and Coarticulation. Phon2: Phonetics Beyond the Basics, 2002, from, <http://www.personal.rdg.ac.uk/~llsroach/phon2/asscoareli-into.htm>
- [7] Jonson R. How to dialogue system compensate for speech recognition deficiencies. Department of Linguistics, Goteborgs University, 2002.
- [8] Phoneme, Wikipedia, from, <http://en.wikipedia.org/wiki/Phoneme>
- [9] Speech Recognition Report, Enginuity, from <http://www-g.eng.cam.ac.uk/enginuity/issue9/article14.html>
- [10] Kirchhoff K. Novel Approaches to Arabic Speech Recognition. Report from the 2002 Johns-Hopkins Summer Workshop, Johns-Hopkins University, 2002.

12 References

Books

Russel, S. and Norvig, P. Artificial Intelligence A Modern Approach. 2nd Ed. New Jersey: Prentice Hall Pearson Education, 2003.

Sommerville I. Software Engineering, 7th Ed. Addison Wesley, 2004

Young, S. The HTK Book version 3.3. Cambridge: Cambridge University Engineering Department, 2005.

Young, S. The ATK Book. Cambridge: Cambridge University Engineering Department, 2005.

Articles and research

Adriaans F., Heukelom M., Koolen M., Lentz T., Rooij O., Vreeswijk D. Building an HMM Speech Recogniser for Dutch. 2004.

Ghamdi M. SAAVB Final Report. King Abdulaziz City of Science and Technology (KACST), 2001.

Jonson R. How to dialogue system compensate for speech recognition deficiencies. Department of Linguistics, Goteborgs University, 2002.

Kirchhoff K. Novel Approaches to Arabic Speech Recognition. Report from the 2002 Johns-Hopkins Summer Workshop, Johns-Hopkins University, 2002.

Moreau N. HTK (v.3.1): Basic Tutorial. Institute for Telecommunication Systems, Technische Universität Berlin, 2002.

Schwitter R. Spoken Language Dialog Systems: Speech Recognition. Macquarie University, 2004.

Web

Speech Recognition Publication (A Joint Project),
MIT Laboratory for Computer Science, Cambridge, Massachusetts, USA
Oregon Graduate Institute of Science & Technology, Portland, Oregon, USA
Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, from
<http://cslu.cse.ogi.edu/HLTSurvey/ch1node4.html>

Creating Office Managed COM Add-Ins with Visual Studio .NET,
MSDN, from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/office06062002.asp>

Speech Recognition Report, Enginuity, from <http://www-g.eng.cam.ac.uk/enginuity/issue9/article14.html>

Typing injuries, Frequently Asked Questions, from <http://www.tifaq.com/archive.html>

Connected Speech and Coarticulation, Phon2: Phonetics Beyond the Basics, 2002, from,
<http://www.personal.rdg.ac.uk/~llsroach/phon2/asscoareli-into.htm>

13 Appendices

Appendix A – HTK Tools

HTK

The Hidden Markov Model Toolkit (HTK) is a general purpose toolkit developed by Cambridge University Engineering Department. It is used for numerous purposes one of which is to facilitate research into the area of speech recognition.

HTK consists of a set of library modules and complex tools available in C source form. These tools provide sophisticated facilities for speech analysis, HMM training, testing and results analysis. Their functions are based on complex scientific, mathematical and statistical knowledge. They run with a command line style.

Standard Tool Options

Options consisting of a capital letter are common across all tools. There are six options that are standard across all tools. The option `-C` is used to specify a configuration file name and the option `-S` is used to specify a script file name, whilst the option `-D` is used to display configuration settings.

Two other standard options are `-A` and `-V`. The option `-A` causes the current command line arguments to be printed. When running experiments via scripts, it is a good idea to use this option to record in a log file the precise settings used for each tool. The option `-V` causes version information for the tool and each module used by that tool to be listed. These should be quoted when making bug reports.

Finally, all tools implement the trace option `-T`. Trace values are typically bit strings. Setting a trace option via the command line overrides any setting for that same trace option in a configuration file. This is a general rule, command line options always override defaults set in configuration files.

As a general rule, passing at least `-A -D -V -T 1` to all tools should be considered, which will guarantee that sufficient information is available in the tool output.

HBuild

Function

This program is used to convert input files that represent language models in a number of different formats and output a standard HTK lattice. The main purpose of HBUILD is to allow the expansion of HTK multi-level lattices and the conversion of bigram language models (such as those generated by HLSTATS) into lattice format.

The specific input file types supported by HBUILD are:

1. HTK multi-level lattice files.
2. Back-off bigram files in ARPA/MIT-LL format.
3. Matrix bigram files produced by HLSTATS.
4. Word lists (to generate a word-loop grammar).
5. Word-pair grammars in ARPA Resource Management format.

Use

HBUILD is invoked by the command line

```
HBuild [options] wordList outLatFile
```

The wordList should contain a list of all the words used in the input language model. The options specify the type of input language model as well as the source filename. If none of the flags specifying input language model type are given a simple word-loop is generated using the wordList given. After processing the input language model, the resulting lattice is saved to file outLatFile.

The operation of HBUILD is controlled by the following command line options:

-b

Output the lattice in binary format. This increases speed of subsequent loading (default ASCII text lattices).

-m fn

The matrix format bigram in fn forms the input language model.

-n fn

The ARPA/MIT-LL format back-off bigram in fn forms the input language model.

-s st en

Set the bigram entry and exit words to st and en. (Default !ENTER and !EXIT). Note that no words will follow the exit word, or precede the entry word. Both the entry and exit word must be included in the wordList. This option is only effective in conjunction with the -n option.

-t st en

This option is used with word-loops and word-pair grammars. An output lattice is produced with an initial word-symbol st (before the loop) and a final word-symbol en (after the loop). This allows initial and final silences to be specified. (Default is that the initial and final nodes are labelled with !NULL). Note that st and en shouldn't be included in the wordList unless they occur elsewhere in the network. This is only effective for word-loop and word-pair grammars.

-u s

The unknown word is s (default !NULL). This option only has an effect when bigram input language models are specified. It can be used in conjunction with the -z flag to delete the symbol for unknown words from the output lattice.

-w fn

The word-pair grammar in fn forms the input language model. The file must be in the format used for the ARPA Resource Management grammar.

-x fn

The extended HTK lattice in fn forms the input language model. This option is used to expand a multi-level lattice into a single level lattice that can be processed by other HTK tools.

-z

Delete (zap) any references to the unknown word (see -u option) in the output lattice.

HCompV

Function

This program will calculate the global mean and covariance of a set of training data. It is primarily used to initialise the parameters of a HMM such that all component means and all covariances are set equal to the global data mean and covariance. This might form the first stage of a flat start training scheme where all models are initially given the same parameters. Alternatively, the covariances may be used as the basis for Fixed Variance and Grand Variance training schemes. These can sometimes be beneficial in adverse conditions where a fixed covariance matrix can give increased robustness.

When training large model sets from limited data, setting a floor is often necessary to prevent variances being badly underestimated through lack of data. One way of doing this is to define a variance macro called `varFloorN` where `N` is the stream index. HCOMPV can also be used to create these variance floor macros with values equal to a specified fraction of the global variance.

Another application of HCOMPV is the estimation of mean and variance vectors for use in cluster-based mean and variance normalisation schemes. Given a list of utterances and a speaker pattern HCOMPV will estimate a mean and a variance for each speaker.

Use

HCOMPV is invoked via the command line

```
HCompV [options] [hmm] trainFiles ...
```

where `hmm` is the name of the physical HMM whose parameters are to be initialised. Note that no HMM name needs to be specified when cepstral mean or variance vectors are estimated (`-c` option). The effect of this command is to compute the covariance of the speech training data and then copy it into every Gaussian component of the given HMM definition. If there are multiple data streams, then a separate covariance is estimated for each stream. The HMM can have a mix of diagonal and full covariances and an option exists to update the means also. The HMM definition can be contained within one or more macro files loaded via the standard `-H` option. Otherwise, the definition will be read from a file called `hmm`. Any tyings in the input definition will be preserved in the output. By default, the new updated definition overwrites the existing one. However, a new

definition file including any macro files can be created by specifying an appropriate target directory using the standard `-M` option.

In addition to the above, an option `-f` is provided to compute variance floor macros equal to a specified fraction of the global variance. In this case, the newly created macros are written to a file called `vFloors`. For each stream `N` defined for `hmm`, a variance macro called `varFloorN` is created. If a target directory is specified using the standard `-M` option then the new file will be written there, otherwise it is written in the current directory.

The list of train files can be stored in a script file if required. Furthermore, the data used for estimating the global covariance can be limited to that corresponding to a specified label.

The calculation of cluster-based mean and variances estimates is enabled by the option `-c` which specifies the output directory where the estimated vectors should be stored.

The detailed operation of HCOMPV is controlled by the following command line options

`-c s`

Calculate cluster-based mean/variance estimate and store results in the specified directory.

`-k s`

Speaker pattern for cluster-based mean/variance estimation. Each utterance filename is matched against the pattern and the characters that are matched against `%` are used as the cluster name. One mean/variance vector is estimated for each cluster.

`-p s`

Path pattern for cluster-based mean/variance estimation. Each utterance filename is matched against the pattern and the characters that are matched against `%` are spliced to the end of the directory string specified with option `'-c'` for the final mean/variance vectors output.

`-q s`

For cluster-based mean/variance estimation different types of output can be requested. Any subset of the letters `nmv` can be specified. Specifying `n` causes the

number of frames in a cluster to be written to the output file. `m` and `v` cause the mean and variance vectors to be included, respectively.

`-f f`

Create variance floor macros with values equal to `f` times the global variance. One macro is created for each input stream and the output is stored in a file called `vFloors`.

`-l s`

The string `s` must be the name of a segment label. When this option is used, HCOMPV searches through all of the training files and uses only the speech frames from segments with the given label. When this option is not used, HCOMPV uses all of the data in each training file.

`-m`

The covariances of the output HMM are always updated however updating the means must be specifically requested. When this option is set, HCOMPV updates all the HMM component means with the sample mean computed from the training files.

`-o s`

The string `s` is used as the name of the output HMM in place of the source name.

`-v f`

This sets the minimum variance (i.e. diagonal elements of the covariance matrix) to the real value `f` (default value 0.0).

`-B`

Output HMM definition files in binary format.

`-F fmt`

Set the source data format to `fmt`.

`-G fmt`

Set the label file format to `fmt`.

`-H mmf`

Load HMM macro model file `mmf`. This option may be repeated to load multiple MMFs.

`-I mlf`

This loads the master label file `mlf`. This option may be repeated to load several MLFs.

`-L dir`

Search directory `dir` for label files (default is to search current directory).

`-M dir`

Store output HMM macro model files in the directory `dir`. If this option is not given, the new HMM definition will overwrite the existing one.

`-X ext`

Set label file extension to `ext` (default is `lab`).

HCopy

Function

This program will copy one or more data files to a designated output file, optionally converting the data into a parameterised form. While the source files can be in any supported format, the output format is always HTK. By default, the whole of the source file is copied to the target but options exist to only copy a specified segment. Hence, this program is used to convert data files in other formats to the HTK format, to concatenate or segment data files, and to parameterise the result. If any option is set which leads to the extraction of a segment of the source file rather than all of it, then segments will be extracted from all source files and concatenated to the target.

Labels will be copied/concatenated if any of the options indicating labels are specified (`-i -l -x -G -I -L -P -X`). In this case, each source data file must have an associated label file, and a target label file is created. The name of the target label file is the root name of the target data file with the extension `.lab`, unless the `-X` option is used. This new label file will contain the appropriately copied/truncated/concatenated labels to correspond with the target data file; all start and end boundaries are recalculated if necessary.

When used in conjunction with HSLAB, HCOPY provides a facility for tasks such as cropping silence surrounding recorded utterances. Since input files may be coerced, HCOPY can also be used to convert the parameter kind of a file, for example from WAVEFORM to MFCC, depending on the configuration options. Conversions must be specified via a configuration file. Note also that the parameterisation qualifier `_N` cannot be used when saving files to disk, and is meant only for on-the-fly parameterisation.

Use

HCOPY is invoked by typing the command line

```
HCopy [options] sa1 [+ sa2 + ...] ta [sb1 [+ sb2 + ...] tb ...]
```

This causes the contents of the one or more source files `sa1`, `sa2`, ... to be concatenated and the result copied to the given target file `ta`. To avoid the overhead of reinvoking the tool when processing large databases, multiple sources and targets may be specified, for example

```
HCopy srcA.wav + srcB.wav tgtAB.wav srcC.wav tgtD.wav
```

will create two new files `tgtAB.wav` and `tgtD.wav`. HCOPY takes file arguments from a script specified using the `-S` option exactly as from the command line, except that any newlines are ignored.

The allowable options to HCOPY are as follows where all times and durations are given in 100 ns units and are written as floating-point numbers.

`-a i`

Use level `i` of associated label files with the `-n` and `-x` options. Note that this is not the same as using the `TRANSLEVEL` configuration variable since the `-a` option still allows all levels to be copied through to the output files.

`-e f`

End copying from the source file at time `f`. The default is the end of the file. If `f` is negative or zero, it is interpreted as a time relative to the end of the file, while a positive value indicates an absolute time from the start of the file.

`-i mlf`

Output label files to master file `mlf`.

- `-l s`
Output label files to the directory `s`. The default is to output to the current directory.
- `-m t`
Set a margin of duration `t` around the segments defined by the `-n` and `-x` options.
- `-n i [j]`
Extract the speech segment corresponding to the `i`'th label in the source file. If `j` is specified, then the segment corresponding to the sequence of labels `i` to `j` is extracted. Labels are numbered from their position in the label file. A negative index can be used to count from the end of the label list. Thus, `-n 1 -1` would specify the segment starting at the first label and ending at the last.
- `-s f`
Start copying from the source file at time `f`. The default is 0.0, ie the beginning of the file.
- `-t n`
Set the line width to `n` chars when formatting trace output.
- `-x s [n]`
Extract the speech segment corresponding to the first occurrence of label `s` in the source file. If `n` is specified, then the `n`'th occurrence is extracted. If multiple files are being concatenated, segments are extracted from each file in turn, and the label must exist for each concatenated file.
- `-F fmt`
Set the source data format to `fmt`.
- `-G fmt`
Set the label file format to `fmt`.
- `-I mlf`
This loads the master label file `mlf`. This option may be repeated to load several MLFs.

`-L dir`

Search directory `dir` for label files (default is to search current directory).

`-O fmt`

Set the target data format to `fmt`.

`-P fmt`

Set the target label format to `fmt`.

`-X ext`

Set label file extension to `ext` (default is `lab`).

HDMAN

Function

The HTK tool HDMAN is used to prepare a pronunciation dictionary from one or more sources. It reads in a list of editing commands from a script file and then outputs an edited and merged copy of one or more dictionaries.

Each source pronunciation dictionary consists of comment lines and definition lines. Comment lines start with the `#` character (or optionally any one of a set of specified comment chars) and are ignored by HDMAN. Each definition line starts with a word and is followed by a sequence of symbols (phones) that define the pronunciation. The words and the phones are delimited by spaces or tabs, and the end of line delimits each definition.

Dictionaries used by HDMAN are read using the standard HTK string conventions, however, the command `IR` can be used in a HDMAN source edit script to switch to using this raw format. Note that in the default mode, words and phones should not begin with unmatched quotes (they should be escaped with the backslash). All dictionary entries must already be alphabetically sorted before using HDMAN.

Each edit command in the script file must be on a separate line. Lines in the script file starting with a `#` are comment lines and are ignored. The commands supported are listed below. They can be displayed by HDMAN using the `-Q` option.

When no edit files are specified, HDMAN simply merges all of the input dictionaries and outputs them in sorted order. All input dictionaries must be sorted. Each input dictionary

xxx may be processed by its own private set of edit commands stored in xxx.ded. Subsequent to the processing of the input dictionaries by their own unique edit scripts, the merged dictionary can be processed by commands in global.ded (or some other specified global edit file name).

Dictionaries are processed on a word by word basis in the order that they appear on the command line. Thus, all of the pronunciations for a given word are loaded into a buffer then all edit commands are applied to these pronunciations. The result is then output and the next word loaded.

Where two or more dictionaries give pronunciations for the same word, the default behaviour is that only the first set of pronunciations encountered are retained and all others are ignored. An option exists to override this so that all pronunciations are concatenated.

Dictionary entries can be filtered by a word list such that all entries not in the list are ignored. Note that the word identifiers in the word list should match exactly (e.g. same case) their corresponding entries in the dictionary.

The edit commands provided by HDMAN are as follows

AS A B ...

Append silence models A, B, etc to each pronunciation.

CR X A Y B

Replace phone Y in the context of A_B by X. Contexts may include an asterix * to denote any phone or a defined context set defined using the DC command.

DC X A B ...

Define the set A, B, ...as the context X.

DD X A B ...

Delete the definition for word X starting with phones A, B,

DP A B C ...

Delete any occurrences of phones A or B or C

DS src

Delete each pronunciation from source src unless it is the only one for the current word.

DW X Y Z ...

Delete words (& definitions) X, Y, Z,

FW X Y Z ...

Define X, Y, Z, ... as function words and change each phone in the definition to a function word specific phone. For example, in word W phone A would become W.A.

IR

Set the input mode to raw. In raw mode, words are regarded as arbitrary sequences of printing chars. In the default mode, words are strings.

LC [X]

Convert all phones to be left-context dependent. If X is given then the 1st phone a in each word is changed to X-a otherwise it is unchanged.

LP

Convert all phones to lowercase.

LW

Convert all words to lowercase.

MP X A B ...

Merge any sequence of phones A B ... and rename as X.

RC [X]

Convert all phones to be right-context dependent. If X is given then the last phone z in each word is changed to z+X otherwise it is unchanged.

RP X A B ...

Replace all occurrences of phones A or B ...by X.

RS system

Remove stress marking. Currently the only stress marking system supported is that used in the dictionaries produced by Carnegie Mellon University (system = cmu).

RW X A B ...

Replace all occurrences of word A or B ...by X.

SP X A B ...

Split phone X into the sequence A B C

TC [X [Y]]

Convert phones to triphones. If X is given then the first phone a is converted to X-a+b otherwise it is unchanged. If Y is given then the last phone z is converted to y-z+Y otherwise if X is given then it is changed to y-z+X otherwise it is unchanged.

UP

Convert all phones to uppercase.

UW

Convert all words to uppercase.

Use

HDMAN is invoked by typing the command line

HDMAN [options] newDict srcDict1 srcDict2 ...

This causes HDMAN read in the source dictionaries srcDict1, srcDict2, etc. and generate a new dictionary newDict. The available options are

-a s

Each character in the string s denotes the start of a comment line. By default there is just one comment character defined which is #.

-b s

Define s to be a word boundary symbol.

-e dir

Look for edit scripts in the directory dir.

-g f

File *f* holds the global edit script. By default, HDMAN expects the global edit script to be called *global.ded*.

-h i j

Skip the first *i* lines of the *j*'th listed source dictionary.

-i

Include word output symbols in the output dictionary.

-j

Include pronunciation probabilities in the output dictionary.

-l s

Write a log file to *s*. The log file will include dictionary statistics and a list of the number of occurrences of each phone.

-m

Merge pronunciations from all source dictionaries. By default, HDMAN generates a single pronunciation for each word. If several input dictionaries have pronunciations for a word, then the first encountered is used. Setting this option causes all distinct pronunciations to be output for each word.

-n f

Output a list of all distinct phones encountered to file *f*.

-o

Disable dictionary output.

-p f

Load the phone list stored in file *f*. This enables a check to be made that all output phones are in the supplied list. You need to create a log file (*-l*) to view the results of this check.

-t

Tag output words with the name of the source dictionary which provided the pronunciation.

-w f

Load the word list stored in file *f*. Only pronunciations for the words in this list will be extracted from the source dictionaries.

-Q

Print a summary of all commands supported by this tool.

HERest

Function

This program is used to perform a single re-estimation of the parameters of a set of HMMs, or linear transforms, using an *embedded training* version of the Baum-Welch algorithm. Training data consists of one or more utterances each of which has a transcription in the form of a standard label file (segment boundaries are ignored). For each training utterance, a composite model is effectively synthesised by concatenating the phoneme models given by the transcription. Each phone model has the same set of accumulators allocated to it as are used in HRest but in HEREST they are updated simultaneously by performing a standard Baum-Welch pass over each training utterance using the composite model.

HEREST is intended to operate on HMMs with initial parameter values estimated by HInit/HRest. HEREST supports multiple mixture Gaussians, discrete and tied-mixture HMMs, multiple data streams, parameter tying within and between models, and full or diagonal covariance matrices. HEREST also supports tee-models for handling optional silence and non-speech sounds. These may be placed between the units (typically words or phones) listed in the transcriptions but they cannot be used at the start or end of a transcription. Furthermore, chains of tee-models are not permitted.

HEREST includes features to allow parallel operation where a network of processors is available. When the training set is large, it can be split into separate chunks that are processed in parallel on multiple machines/processors, consequently speeding up the training process.

Like all re-estimation tools, HEREST allows a floor to be set on each individual variance by defining a variance floor macro for each data stream. The configuration variable VARFLOORPERCENTILE allows the same thing to be done in a different way which appears to improve recognition results. By setting this to e.g. 20, the variances from each dimension are floored to the 20th percentile of the distribution of variances for that dimension.

HEREST supports two specific methods for initialisation of model parameters, single pass retraining and 2-model re-estimation.

Single pass retraining is useful when the parameterisation of the front-end (e.g. from MFCC to PLP coefficients) is to be modified. Given a set of well-trained models, a set of new models using a different parameterisation of the training data can be generated in a single pass. This is done by computing the forward and backward probabilities using the original well-trained models and the original training data, but then switching to a new set of training data to compute the new parameter estimates.

In 2-model re-estimation one model set can be used to obtain the forward backward probabilities which then are used to update the parameters of another model set. Contrary to single pass retraining the two model sets are not required to be tied in the same fashion. This is particularly useful for training of single mixture models prior to decision-tree based state clustering. The use of 2-model re-estimation in HEREST is triggered by setting the config variables `ALIGNMODELMMF` or `ALIGNMODELDIR` and `ALIGNMODELEXT` together with `ALIGNHMMMLIST`. As the model list can differ for the alignment model set a separate set of input transforms may be specified using the `ALIGNXFORMDIR` and `ALIGNXFORMEXT`.

HEREST for updating model parameters operates in two distinct stages.

1. In the first stage, one of the following two options applies
 - a. Each input data file contains training data which is processed and the accumulators for state occupation, state transition, means and variances are updated.
 - b. Each data file contains a dump of the accumulators produced by previous runs of the program. These are read in and added together to form a single set of accumulators.
2. In the second stage, one of the following options applies
 - a. The accumulators are used to calculate new estimates for the HMM parameters.
 - b. The accumulators are dumped into a file.

Thus, on a single processor the default combination 1(a) and 2(a) would be used. However, if N processors are available then the training data would be split into N equal

groups and HEREST would be set to process one data set on each processor using the combination 1(a) and 2(b). When all processors had finished, the program would then be run again using the combination 1(b) and 2(a) to load in the partial accumulators created by the N processors and do the final parameter re-estimation. The choice of which combination of operations HEREST will perform is governed by the `-p` option switch as described below.

As a further performance optimisation, HEREST will also prune the α and β matrices. By this means, a factor of 3 to 5 speed improvement and a similar reduction in memory requirements can be achieved with negligible effects on training performance (see the `-t` option below).

HEREST is able to make use of, and estimate, linear transformations for model adaptation. There are three types of linear transform that are made use in HEREST.

- *Input transform*: the input transform is used to determine the forward-backward probabilities, hence the component posteriors, for estimating model and transform.
- *Output transform*: the output transform is generated when the `-u` option is set to a. The transform will be stored in the current directory, or the directory specified by the `-K` option and optionally the transform extension.
- *Parent transform*: the parent transform determines the model, or features, on which the model set or transform is to be generated. For transform estimation this allows *cascades* of transforms to be used to adapt the model parameters. For model estimation this supports *speaker adaptive training*. Note the current implementation only supports adaptive training with CMLLR. Any parent transform can be used when generating transforms.

When input or parent transforms are specified the transforms may physically be stored in multiple directories. Which transform to be used is determined in the following search order: order is used.

1. Any loaded macro that matches the transform (and its' extension) name.
2. If it is a parent transform, the directory specified with the `-E` option.
3. The list of directories specified with the `-J` option. The directories are searched in the order that they are specified in the command line.

As the search order above looks for loaded macros first it is recommended that unique extensions are specified for each set of transforms generated. Transforms may either be stored in a single TMF. These TMFs may be loaded using the -H option. When macros are specified for the regression class trees and the base classes the following search order is used

1. Any loaded macro that matches the macro name.
2. The path specified by the configuration variable.
3. The list of directories specified with the -J option. The directories are searched in the order that they are specified in the command line.

Base classes and regression classes may also be loaded using the -H option.

Use

HEREST is invoked via the command line

```
HERest [options] hmmList trainFile ...
```

This causes the set of HMMs given in `hmmList` to be loaded. The given list of training files is then used to perform one re-estimation cycle. As always, the list of training files can be stored in a script file if required. On completion, HEREST outputs new updated versions of each HMM definition. If the number of training examples falls below a specified threshold for some particular HMM, then the new parameters for that HMM are ignored and the original parameters are used instead.

The detailed operation of HEREST is controlled by the following command line options

-a

Use an input transform to obtain alignments for updating models or transforms (default off).

-c f

Set the threshold for tied-mixture observation pruning to f. For tied-mixture TIEDHS systems, only those mixture component probabilities which fall within f of the maximum mixture component probability are used in calculating the state output probabilities (default 10.0).

-d dir

Normally HEREST looks for HMM definitions (not already loaded via MMF files) in the current directory. This option tells HEREST to look in the directory `dir` to find them.

`-h mask`

Set the mask for determining which transform names are to be used for the output transforms. If `PAXFORMMASK` or `INXFORMMASK` are not specified then the input transform mask is assumed for both output and parent transforms.

`-l N`

Set the maximum number of files to use for each speaker, determined by the output transform speaker mask, to estimate the transform with. (default `∞`).

`-m N`

Set the minimum number of training examples required for any model to `N`. If the actual number falls below this value, the HMM is not updated and the original parameters are used for the new version (default value 3).

`-o ext`

This causes the file name extensions of the original models (if any) to be replaced by `ext`.

`-p N`

This switch is used to set parallel mode operation. If `p` is set to a positive integer `N`, then HEREST will process the training files and then dump all the accumulators into a file called `HERN.acc`. If `p` is set to 0, then it treats all file names input on the command line as the names of `.acc` dump files. It reads them all in, adds together all the partial accumulations and then re-estimates all the HMM parameters in the normal way.

`-r`

This enables single-pass retraining. The list of training files is processed pair-by-pair. For each pair, the first file should match the parameterisation of the original model set. The second file should match the parameterisation of the required new set. All speech input processing is controlled by configuration variables in the normal way except that the variables describing the old parameterisation are qualified by the name `HPARM1` and the variables describing the new

parameterisation are qualified by the name HPARM2. The stream widths for the old and the new must be identical.

-s file

This causes statistics on occupation of each state to be output to the named file. This file is needed for the RO command of HHed but it is also generally useful for assessing the amount of training material available for each HMM state.

-t f [i l]

Set the pruning threshold to f. During the backward probability calculation, at each time t all (log) β values falling more than f below the maximum β value at that time are ignored. During the subsequent forward pass, (log) α values are only calculated if there are corresponding valid β values. Furthermore, if the ratio of the $\alpha \beta$ product divided by the total probability (as computed on the backward pass) falls below a fixed threshold then those values of α and β are ignored. Setting f to zero disables pruning (default value 0.0). Tight pruning thresholds can result in HEREST failing to process an utterance. If the i and l options are given, then a pruning error results in the threshold being increased by i and utterance processing restarts. If errors continue, this procedure will be repeated until the limit l is reached.

-u flags

By default, HEREST updates all of the HMM parameters, that is, means, variances, mixture weights and transition probabilities. This option causes just the parameters indicated by the flags argument to be updated, this argument is a string containing one or more of the letters m (mean), v (variance), t (transition), a (linear transform), p (use MAP adaptation), and w (mixture weight). The presence of a letter enables the updating of the corresponding parameter set.

-v f

This sets the minimum variance (i.e. diagonal element of the covariance matrix) to the real value f (default value 0.0).

-w f

Any mixture weight which falls below the global constant MINMIX is treated as being zero. When this parameter is set, all mixture weights are floored to f * MINMIX.

-x ext

By default, HEREST expects a HMM definition for the label X to be stored in a file called X. This option causes HEREST to look for the HMM definition in the file X.ext.

-z file

Save all output transforms to file. Default is TMF.

-B

Output HMM definition files in binary format.

-E dir [ext]

Parent transform directory and optional extension for parent transforms. The default option is that no parent transform is used.

-F fmt

Set the source data format to fmt.

-G fmt

Set the label file format to fmt.

-H mmf

Load HMM macro model file mmf. This option may be repeated to load multiple MMFs.

-I mlf

This loads the master label file mlf. This option may be repeated to load several MLFs.

-J dir [ext]

Add directory to the list of possible input transform directories. Only one of the options can specify the extension to use for the input transforms.

-K dir [ext]

Output transform directory and optional extension for output transforms. The default option is that there is no output extension and the current transform directory is used.

-L dir

Search directory dir for label files (default is to search current directory).

-M dir

Store output HMM macro model files in the directory dir. If this option is not given, the new HMM definition will overwrite the existing one.

-X ext

Set label file extension to ext (default is lab).

HHEd

Function

HHED is a script driven editor for manipulating sets of HMM definitions. Its basic operation is to load in a set of HMMs, apply a sequence of edit operations and then output the transformed set. HHED is mainly used for applying tyings across selected HMM parameters. It also has facilities for cloning HMMs, clustering states and editing HMM structures.

Many HHED commands operate on sets of similar items selected from the set of currently loaded HMMs. For example, it is possible to define a set of all final states of all vowel models, or all mean vectors of all mixture components within the model X, etc.

Use

HHED is invoked by typing the command line

```
HHEd [options] edCmdFile hmmList
```

where edCmdFile is a text file containing a sequence of edit commands as described above and hmmList defines the set of HMMs to be edited (see HMODEL for the format of HMM list). If the models are to be kept in separate files rather than being stored in an MMF, the configuration variable KEEPDISTINCT should be set to true. The available options for HHED are

-d dir

This option tells HHED to look in the directory dir to find the model definitions.

-o ext

This causes the file name extensions of the original models (if any) to be replaced by ext.

-w mmf

Save all the macros and model definitions in a single master macro file mmf.

-x s

Set the extension for the edited output files to be s (default is to use the original names unchanged).

-z

Setting this option causes all aliases in the loaded HMM set to be deleted (zapped) immediately before loading the definitions. The result is that all logical names are ignored and the actual HMM list consists of just the physically distinct HMMs.

-B

Output HMM definition files in binary format.

-H mmf

Load HMM macro model file mmf. This option may be repeated to load multiple MMFs.

-M dir

Store output HMM macro model files in the directory dir. If this option is not given, the new HMM definition will overwrite the existing one.

-Q

Print a summary of all commands supported by this tool.

HLStats

Function

This program will read in a HMM list and a set of HTK format transcriptions (label files). It will then compute various statistics which are intended to assist in analysing acoustic training data and generating simple language models for recognition. The specific functions provided by HLSTATS are:

1. number of occurrences of each distinct logical HMM and/or each distinct physical HMM. The list printed can be limited to the N most infrequent models.
2. minimum, maximum and average durations of each logical HMM in the transcriptions.
3. a matrix of bigram probabilities
4. an ARPA/MIT-LL format text file of back-off bigram probabilities
5. a list of labels which cover the given set of transcriptions.

Bigram Generation

When using the bigram generating options, each transcription is assumed to have a unique entry and exit label which by default are !ENTER and !EXIT. If these labels are not present they are inserted. In addition, any label occurring in a transcription which is not listed in the HMM list is mapped to a unique label called !NULL.

HLSTATS processes all input transcriptions and maps all labels to a set of unique integers in the range 1 to L , where L is the number of distinct labels. For each adjacent pair of labels i and j , it counts the total number of occurrences $N(i, j)$. Let the total number

$$N(i) = \sum_{j=1}^L N(i, j)$$

of occurrences of label i be

For matrix bigrams, the bigram probability $p(i, j)$ is given by

$$p(i, j) = \begin{cases} \alpha N(i, j) / N(i) & \text{if } N(i) > 0 \\ 1/L & \text{if } N(i) = 0 \\ f & \text{otherwise} \end{cases}$$

where f is a floor probability set by the -f option and α is chosen to ensure that

$$\sum_{j=1}^L p(i, j) = 1$$

For back-off bigrams, the unigram probabilities $p(i)$ are given by

$$p(i) = \begin{cases} N(i)/N & \text{if } N(i) > u \\ u/N & \text{otherwise} \end{cases}$$

where u is unigram floor count set by the -u option and $N = \sum_{i=1}^L \max[N(i), u]$.

The backed-off bigram probabilities are given by

$$p(i, j) = \begin{cases} (N(i, j) - D)/N(i) & \text{if } N(i, j) > t \\ b(i)p(j) & \text{otherwise} \end{cases}$$

where D is a discount and t is a bigram count threshold set by the -t option. The discount D is fixed at 0.5 but can be changed via the configuration variable DISCOUNT. The

back-off weight $b(i)$ is calculated to ensure that $\sum_{j=1}^L p(i, j) = 1$, i.e.

$$b(i) = \frac{1 - \sum_{j \in B} p(i, j)}{1 - \sum_{j \in B} p(j)}$$

where B is the set of all words for which $p(i, j)$ has a bigram.

Use

HLSTATS is invoked by the command line

HLStats [options] hmmList labFiles

The hmmList should contain a list of all the labels (ie model names) used in the following label files for which statistics are required. Any labels not appearing in the list are ignored and assumed to be out-of-vocabulary. The list of labels is specified in the same way as for a HMM list (see HMODEL) and the logical \Rightarrow physical mapping is preserved to allow statistics to be collected about physical names as well as logical ones. The labFiles may be master label files. The available options are

-b fn

Compute bigram statistics and store result in the file fn.

- c N**
Count the number of occurrences of each logical model listed in the `hmmList` and on completion list all models for which there are N or less occurrences.
- d**
Compute minimum, maximum and average duration statistics for each label.
- f f**
Set the matrix bigram floor probability to f (default value 0.0). This option should be used in conjunction with the `-b` option.
- h N**
Set the bigram hashtable size to medium(N=1) or large (N=2). This option should be used in conjunction with the `-b` option. The default is small(N=0).
- l fn**
Output a list of covering labels to file `fn`. Only labels occurring in the `labList` are counted (others are assumed to be out-of-vocabulary). However, this list may contain labels that do not occur in any of the label files. The list of labels written to `fn` will however contain only those labels which occur at least once.
- o**
Produce backed-off bigrams rather than matrix ones. These are output in the standard ARPA/MIT-LL textual format.
- p N**
Count the number of occurrences of each physical model listed in the `hmmList` and on completion list all models for which there are N or less occurrences.
- s st en**
Set the sentence start and end labels to `st` and `en`. (Default !ENTER and !EXIT).
- t n**
Set the threshold count for including a bigram in a backed-off bigram language model. This option should be used in conjunction with the `-b` and `-o` options.
- u f**

Set the unigram floor probability to *f* when constructing a back-off bigram language model. This option should be used in conjunction with the *-b* and *-o* options.

-G fmt

Set the label file format to *fmt*.

-I mlf

This loads the master label file *mlf*. This option may be repeated to load several MLFs.

HParse

Function

The HPARSE program generates word level lattice files (for use with e.g. HVITE) from a text file syntax description containing a set of rewrite rules based on extended Backus-Naur Form (EBNF). The EBNF rules are used to generate an internal representation of the corresponding finite-state network where HPARSE network nodes represent the words in the network, and are connected via sets of links. This HPARSE network is then converted to HTK V2 word level lattice. The program provides one convenient way of defining such word level lattices.

HPARSE also provides a *compatibility mode* for use with HPARSE syntax descriptions used in HTK V1.5 where the same format was used to define both the word level syntax and the dictionary. In compatibility mode HPARSE will output the word level portion of such a syntax as an HTK V2 lattice file (via HNET) and the pronunciation information as an HTK V2 dictionary file (via HDICT).

The lattice produced by HPARSE will often contain a number of !NULL nodes in order to reduce the number of arcs in the lattice. The use of such !NULL nodes can both reduce size and increase efficiency when used by recognition programs such as HVITE.

Network Definition

The syntax rules for the textual definition of the network are as follows. Each node in the network has a nodename. This node name will normally correspond to a word in the final syntax network. Additionally, for use in compatibility mode, each node can also have an external name.

name = char{char}

`nodename = name ["%" ("%" | name)]`

Here char represents any character except one of the meta chars { } [] < > | = \$ (); \ / *. The latter may, however, be escaped using a backslash. The first name in a nodename represents the name of the node (``internal name"), and the second optional name is the ``external" name. This is used only in compatibility mode, and is, by default the same as the internal name.

Network definitions may also contain variables

`variable = $name`

Variables are identified by a leading \$ character. They stand for sub-networks and must be defined before they appear in the RHS of a rule using the form

`subnet = variable "=" expr ";"`

An expr consists of a set of alternative sequences representing parallel branches of the network.

`expr = sequence { "|" sequence }`

`sequence = factor { factor }`

Each sequence is composed of a sequence of factors where a factor is either a node name, a variable representing some sub-network or an expression contained within various sorts of brackets.

`factor = "{ expr }" |`
 "`{` expr `}`" |
 "< expr >" |
 "[expr]" |
 "<< expr >>" |
 nodename | variable

Ordinary parentheses are used to bracket sub-expressions, curly braces { } denote zero or more repetitions and angle brackets < > denote one or more repetitions. Square brackets [] are used to enclose optional items. The double angle brackets are a special feature

included for building context dependent loops and are explained further below. Finally, the complete network is defined by a list of sub-network definitions followed by a single expression within parentheses.

```
network = {subnet} "{" expr "}"
```

Note that C style comments may be placed anywhere in the text of the network definition.

As an example, the following network defines a syntax for some simple edit commands

```
$dir = up | down | left | right;

$mvcmd = move $dir | top | bottom;

$item = char | word | line | page;

$dldcmd = delete [$item]; /* default is char */

$incmd = insert;

$encmd = end [insert];

$cmd = $mvcmd|$dldcmd|$incmd|$encmd;

({sil} < $cmd {sil} > quit)
```

Double angle brackets are used to construct contextually consistent context-dependent loops such as a word-pair grammar. This function can also be used to generate consistent triphone loops for phone recognition. The entry and exit conditions to a context-dependent loop can be controlled by the invisible pseudo-words TLOOP_BEGIN and TLOOP_END. The right context of TLOOP_BEGIN defines the legal loop start nodes, and the left context of TLOOP_END defines the legal loop finishers. If TLOOP_BEGIN/TLOOP_END are not present then all models are connected to the entry/exit of the loop.

A word-pair grammar simply defines the legal set of words that can follow each word in the vocabulary. To generate a network to represent such a grammar a right context-dependent loop could be used. The legal sentence set of sentence start and end words are defined as above using TLOOP_BEGIN/TLOOP_END.

For example, the following lists the legal followers for each word in a 7 word vocabulary

ENTRY - show, tell, give

show - me, all

tell - me, all

me - all

all - names, addresses

names - and, names, addresses, show, tell, EXIT

addresses - and, names, addresses, show, tell, EXIT

and - names, addresses, show, tell

HPARSE can generate a suitable lattice to represent this word-pair grammar by using the following specification:

\$TLOOP_BEGIN_FLLWRS = show|tell|give;

\$TLOOP_END_PREDS = names|addresses;

\$show_FLLWRS = me|all;

\$tell_FLLWRS = me|all;

\$me_FLLWRS = all;

\$all_FLLWRS = names|addresses;

\$names_FLLWRS = and|names|addresses|show|tell|TLOOP_END;

\$addresses_FLLWRS = and|names|addresses|show|tell|TLOOP_END;

\$and_FLLWRS = names|addresses|show|tell;

(sil <<

TLOOP_BEGIN+TLOOP_BEGIN_FLLWRS |

TLOOP_END_PREDS-TLOOP_END |

show+show_FLLWRS |

```

tell+tell_FLLWRS |

me+me_FLLWRS |

all+all_FLLWRS |

names+names_FLLWRS |

addresses+addresses_FLLWRS |

and+and_FLLWRS

>> sil )

```

where it is assumed that each utterance begins and ends with sil model.

In this example, each set of contexts is defined by creating a variable whose alternatives are the individual contexts. The actual context-dependent loop is indicated by the « » brackets. Each element in this loop is a single variable name of the form A-B+C where A represents the left context, C represents the right context and B is the actual word. Each of A, B and C can be nodenames or variable names but note that this is the only case where variable names are expanded automatically and the usual \$ symbol is not used. Both A and C are optional, and left and right contexts can be mixed in the same triphone loop.

Compatibility Mode

In HPARSE compatibility mode, the interpretation of the ENBF network is that used by the HTK V1.5 HVITE program. In which HPARSE ENBF notation was used to define both the word level syntax and the dictionary. Compatibility mode is aimed at converting files written for HTK V1.5 into their equivalent HTK V2 representation. Therefore HPARSE will output the word level portion of such a ENBF syntax as an HTK V2 lattice file and the pronunciation information is optionally stored in an HTK V2 dictionary file. When operating in compatibility mode and not generating dictionary output, the pronunciation information is discarded.

In compatibility mode, the reserved node names WD_BEGIN and WD_END are used to delimit word boundaries--nodes between a WD_BEGIN/WD_END pair are called ``word-internal" while all other nodes are ``word-external". All WD_BEGIN/WD_END nodes must have an ``external name" attached that denotes the word. It is a requirement that the number of WD_BEGIN and the number of WD_END nodes are equal and

furthermore that there isn't a direct connection from a WD_BEGIN node to a WD_END. For example a portion of such an HTK V1.5 network could be

```
$A      = WD_BEGIN%A ax WD_END%A;

$ABDOMEN = WD_BEGIN%ABDOMEN ae b d ax m ax n WD_END%ABDOMEN;

$ABIDES  = WD_BEGIN%ABIDES ax b ay d z WD_END%ABIDES;

$ABOLISH = WD_BEGIN%ABOLISH ax b aa l ih sh WD_END%ABOLISH;

... etc

( <

    $A | $ABDOMEN | $ABIDES | $ABOLISH | ... etc

> )
```

HPARSE will output the connectivity of the words in an HTK V2 word lattice format file and the pronunciation information in an HTK V2 dictionary. Word-external nodes are treated as words and stored in the lattice with corresponding entries in the dictionary.

It should be noted that in HTK V1.5 any ENBF network could appear between a WD_BEGIN/WD_END pair, which includes loops. Care should therefore be taken with syntaxes that define very complex sets of alternative pronunciations. It should also be noted that each dictionary entry is limited in length to 100 phones. If multiple instances of the same word are found in the expanded HParse network, a dictionary entry will be created for only the first instance and subsequent instances are ignored (a warning is printed). If words with a NULL external name are present then the dictionary will contain a NULL output symbol.

Use

HPARSE is invoked via the command line

```
HParse [options] syntaxFile latFile
```

HPARSE will then read the set of ENBF rules in syntaxFile and produce the output lattice in latFile.

The detailed operation of HPARSE is controlled by the following command line options

-b

Output the lattice in binary format. This increases speed of subsequent loading (default ASCII text lattices).

-c

Set V1.5 compatibility mode. Compatibility mode can also be set by using the configuration variable V1COMPAT (default compatibility mode disabled).

-d s

Output dictionary to file s. This is only a valid option when operating in compatibility mode. If not set no dictionary will be produced.

-l

Include language model log probabilities in the output These log probabilities are calculated as $-\log(\text{number of followers})$ for each network node.

HRest

Function

HREST performs basic Baum-Welch re-estimation of the parameters of a single HMM using a set of observation sequences. HREST can be used for normal isolated word training in which the observation sequences are realisations of the corresponding vocabulary word.

Alternatively, HREST can be used to generate seed HMMs for phoneme-based recognition. In this latter case, the observation sequences will consist of segments of continuously spoken training material. HREST will cut these out of the training data automatically by simply giving it a segment label.

In both of the above applications, HREST is intended to operate on HMMs with initial parameter values estimated by HINIT.

HREST supports multiple mixture components, multiple streams, parameter tying within a single model, full or diagonal covariance matrices, tied-mixture models and discrete models. The outputs of HREST are often further processed by HEREST.

Like all re-estimation tools, HREST allows a floor to be set on each individual variance by defining a variance floor macro for each data stream. If any diagonal covariance component falls below 0.00001, then the corresponding mixture weight is set to zero. A warning is issued if the number of mixtures is greater than one, otherwise an error

occurs. Applying a variance floor via the -v option or a variance floor macro can be used to prevent this.

Use

HREST is invoked via the command line

```
HRest [options] hmm trainFiles ...
```

This causes the parameters of the given hmm to be re-estimated repeatedly using the data in trainFiles until either a maximum iteration limit is reached or the re-estimation converges. The HMM definition can be contained within one or more macro files loaded via the standard -H option. Otherwise, the definition will be read from a file called hmm. The list of train files can be stored in a script file if required.

The detailed operation of HREST is controlled by the following command line options

-c f

Set the threshold for tied-mixture observation pruning to f. When all mixtures of all models are tied to create a full tied-mixture system, the calculation of output probabilities is treated as a special case. Only those mixture component probabilities which fall within f of the maximum mixture component probability are used in calculating the state output probabilities (default 10.0).

-e f

This sets the convergence factor to the real value f. The convergence factor is the relative change between successive values of $P(O|\lambda)$ (default value 0.0001).

-i N

This sets the maximum number of re-estimation cycles to N (default value 20).

-l s

The string s must be the name of a segment label. When this option is used, HREST searches through all of the training files and cuts out all segments with the given label. When this option is not used, HREST assumes that each training file is a single token.

-m N

Sets the minimum number of training examples to be N. If fewer than N examples are supplied then an error is reported (default value 3).

-t

Normally, training sequences are rejected if they have fewer frames than the number of emitting states in the HMM. Setting this switch disables this reject mechanism.

-u flags

By default, HREST updates all of the HMM parameters, that is, means, variances, mixture weights and transition probabilities. This option causes just the parameters indicated by the flags argument to be updated, this argument is a string containing one or more of the letters m (mean), v (variance), t (transition) and w (mixture weight). The presence of a letter enables the updating of the corresponding parameter set.

-v f

This sets the minimum variance (i.e. diagonal element of the covariance matrix) to the real value f. This is ignored if an explicit variance floor macro is defined. The default value is 0.0.

-w f

Any mixture weight or discrete observation probability which falls below the global constant MINMIX is treated as being zero. When this parameter is set, all mixture weights are floored to $f * \text{MINMIX}$.

-B

Output HMM definition files in binary format.

-F fmt

Set the source data format to fmt.

-G fmt

Set the label file format to fmt.

-H mmf

Load HMM macro model file mmf. This option may be repeated to load multiple MMFs.

-I mlf

This loads the master label file mlf. This option may be repeated to load several MLFs.

-L dir

Search directory dir for label files (default is to search current directory).

-M dir

Store output HMM macro model files in the directory dir. If this option is not given, the new HMM definition will overwrite the existing one.

-X ext

Set label file extension to ext (default is lab).

HResults

Function

HRESULTS is the HTK performance analysis tool. It reads in a set of label files (typically output from a recognition tool such as HVITE) and compares them with the corresponding reference transcription files. For the analysis of speech recognition output, the comparison is based on a Dynamic Programming-based string alignment procedure. For the analysis of word-spotting output, the comparison uses the standard US NIST FOM metric.

When used to calculate the sentence accuracy using DP the basic output is recognition statistics for the whole file set in the format

----- Overall Results -----

SENT: %Correct=13.00 [H=13, S=87, N=100]

WORD: %Corr=53.36, Acc=44.90 [H=460,D=49,S=353,I=73,N=862]

=====

The first line gives the sentence-level accuracy based on the total number of label files which are identical to the transcription files. The second line is the word accuracy based on the DP matches between the label files and the transcriptions. In this second line, **H** is the number of correct labels, **D** is the number of deletions, **S** is the number of substitutions, **I** is the number of insertions and **N** is the total number of labels in the

defining transcription files. The percentage number of labels correctly recognised is given by

$$\text{\%Correct} = \frac{H}{N} \times 100\%$$

and the accuracy is computed by

$$\text{Accuracy} = \frac{H - I}{N} \times 100\%$$

In addition to the standard HTK output format, HRESULTS provides an alternative similar to that used in the US NIST scoring package, i.e.

```
|=====|
|      # Snt | Corr  Sub  Del  Ins  Err S. Err | |
|---|---|---|
| Sum/Avg | 87 | 53.36 40.95 5.68 8.47 55.10 87.00 |
|-----|
```

When HRESULTS is used to generate a confusion matrix, the values are as follows:

%c

The percentage correct in the row; that is, how many times a phone instance was correctly labelled.

%e

The percentage of incorrectly labeled phones in the row as a percentage of the total number of labels in the set.

An example from the HTKDemo routines:

```
===== HTK Results Analysis =====
```

Date: Thu Jan 10 19:00:03 2002

Ref : labels/bcplabs/mon

Rec : test/te1.rec

: test/te2.rec

: test/te3.rec

----- Overall Results -----

SENT: %Correct=0.00 [H=0, S=3, N=3]

WORD: %Corr=63.91, Acc=59.40 [H=85, D=35, S=13, I=6, N=133]

----- Confusion Matrix -----

S C V N L Del [%c / %e]

S 6 1 0 1 0 0 [75.0/1.5]

C 2 35 3 1 0 18 [85.4/4.5]

V 0 1 28 0 1 12 [93.3/1.5]

N 0 1 0 7 0 1 [87.5/0.8]

L 0 1 1 0 9 4 [81.8/1.5]

Ins 2 2 0 2 0

=====

Reading across the rows, %c indicates the number of correct instances divided by the total number of instances in the row. %e is the number of incorrect instances in the row divided by the total number of instances (N).

Optional extra outputs available from HRESULTS are

- recognition statistics on a per file basis
- recognition statistics on a per speaker basis

- recognition statistics from best of N alternatives
- time-aligned transcriptions
- confusion matrices

For comparison purposes, it is also possible to assign two labels to the same equivalence class (see -e option). Also, the *null* label ??? is defined so that making any label equivalent to the null label means that it will be ignored in the matching process. Note that the order of equivalence labels is important, to ensure that label X is ignored, the command line option -e ??? X would be used. Label files containing triphone labels of the form A-B+C can be optionally stripped down to just the class name B via the -s switch.

The word spotting mode of scoring can be used to calculate hits, false alarms and the associated figure of merit for each of a set of keywords. Optionally it can also calculate ROC information over a range of false alarm rates. A typical output is as follows

----- Figures of Merit -----

KeyWord:	#Hits	#FAs	#Actual	FOM
----------	-------	------	---------	-----

A:	8	1	14	30.54
----	---	---	----	-------

B:	4	2	14	15.27
----	---	---	----	-------

Overall:	12	3	28	22.91
----------	----	---	----	-------

which shows the number of hits and false alarms (FA) for two keywords A and B. A label in the test file with start time t_s and end time t_e constitutes a hit if there is a corresponding label in the reference file such that $t_s < t_m < t_e$ where t_m is the mid-point of the reference label.

Note that for keyword scoring, the test transcriptions must include a score with each labelled word spot and all transcriptions must include boundary time information.

The FOM gives the % of hits averaged over the range 1 to 10 FA's per hour. This is calculated by first ordering all spots for a particular keyword according to the match

score. Then for each FA rate f , the number of hits are counted starting from the top of the ordered list and stopping when f have been encountered. This corresponds to a posteriori setting of the keyword detection threshold and effectively gives an upper bound on keyword spotting performance.

Use

HRESULTS is invoked by typing the command line

```
HResults [options] hmmList recFiles ...
```

This causes HRESULTS to be applied to each recFile in turn. The hmmList should contain a list of all model names for which result information is required. Note, however, that since the context dependent parts of a label can be stripped, this list is not necessarily the same as the one used to perform the actual recognition. For each recFile, a transcription file with the same name but the extension .lab (or some user specified extension - see the -X option) is read in and matched with it. The recfiles may be master label files (MLFs), but note that even if such an MLF is loaded using the -I option, the list of files to be checked still needs to be passed, either as individual command line arguments or via a script with the -S option. For this reason, it is simpler to pass the recFile MLF as one of the command line filename arguments. For loading reference label file MLF's, the -I option must be used. The reference labels and the recognition labels must have different file extensions. The available options are

-a s

change the label SENT in the output to s.

-b s

change the label WORD in the output to s.

-c

when comparing labels convert to upper case. Note that case is still significant for equivalences (see -e below).

-d N

search the first N alternatives for each test label file to find the most accurate match with the reference labels. Output results will be based on the most accurate match to allow NBest error rates to be found.

-e s t

the label t is made equivalent to the label s. More precisely, t is assigned to an equivalence class of which s is the identifying member.

-f

Normally, HRESULTS accumulates statistics for all input files and just outputs a summary on completion. This option forces match statistics to be output for each input test file.

-g fmt

This sets the test label format to fmt. If this is not set, the recFiles should be in the same format as the reference files.

-h

Output the results in the same format as US NIST scoring software.

-k s

Collect and output results on a speaker by speaker basis (as well as globally). s defines a pattern which is used to extract the speaker identifier from the test label file name. In addition to the pattern matching metacharacters * and ? (which match zero or more characters and a single character respectively), the character % matches any character whilst including it as part of the speaker identifier.

-m N

Terminate after collecting statistics from the first N files.

-n

Set US NIST scoring software compatibility.

-p

This option causes a phoneme confusion matrix to be output.

-s

This option causes all phoneme labels with the form A-B+C to be converted to B. It is useful for analysing the results of phone recognisers using context dependent models.

-t

This option causes a time-aligned transcription of each test file to be output provided that it differs from the reference transcription file.

-u f

Changes the time unit for calculating false alarm rates (for word spotting scoring) to f hours (default is 1.0).

-w

Perform word spotting analysis rather than string accuracy calculation.

-z s

This redefines the null class name to s. The default null class name is ???, which may be difficult to manage in shell script programming.

-G fmt

Set the label file format to fmt.

-I mlf

This loads the master label file mlf. This option may be repeated to load several MLFs.

-L dir

Search directory dir for label files (default is to search current directory).

-X ext

Set label file extension to ext (default is lab).

HVite

Function

HVITE is a general-purpose Viterbi word recogniser. It will match a speech file against a network of HMMs and output a transcription for each. When performing N-best recognition a word level lattice containing multiple hypotheses can also be produced.

Either a word level lattice or a label file is read in and then expanded using the supplied dictionary to create a model based network. This allows arbitrary finite state word networks and simple forced alignment to be specified.

This expansion can be used to create context independent, word internal context dependent and cross word context dependent networks. The way in which the expansion is performed is determined automatically from the dictionary and HMMList. When all labels appearing in the dictionary are defined in the HMMList no expansion of model names is performed. Otherwise if all the labels in the dictionary can be satisfied by models dependent only upon word internal context these will be used else cross word context expansion will be performed. These defaults can be overridden by HNET configuration parameters.

HVITE supports shared parameters and appropriately pre-computes output probabilities. For increased processing speed, HVITE can optionally perform a beam search controlled by a user specified threshold (see -t option). When fully tied mixture models are used, observation pruning is also provided (see the -c option). Speaker adaptation is also supported by HVITE both in terms of recognition using an adapted model set or a TMF (see the -k option), and in the estimation of a transform by unsupervised adaptation using linear transformation in an incremental mode (see the -j option) or in a batch mode (-K option).

Use

HVITE is invoked via the command line

HVite [options] dictFile hmmList testFiles ...

HVite will then either load a single network file and match this against each of the test files -w netFile, or create a new network for each test file either from the corresponding label file -a or from a word lattice -w. When a new network is created for each test file the path name of the label (or lattice) file to load is determined from the test file name and the -L and -X options described below.

If no testFiles are specified the -w s option must be specified and recognition will be performed from direct audio.

The hmmList should contain a list of the models required to construct the network from the word level representation.

The recogniser output is written in the form of a label file whose path name is determined from the test file name and the -l and -x options described below. The list of test files can be stored in a script file if required.

When performing N-best recognition (see -n N option described below) the output label file can contain multiple alternatives -n N M and a lattice file containing multiple hypotheses can be produced.

The detailed operation of HVITE is controlled by the following command line options

-a

Perform alignment. HVITE will load a label file and create an alignment network for each test file.

-b s

Use s as the sentence boundary during alignment.

-c f

Set the tied-mixture observation pruning threshold to f. When all mixtures of all models are tied to create a full tied-mixture system, the calculation of output probabilities is treated as a special case. Only those mixture component probabilities which fall within f of the maximum mixture component probability are used in calculating the state output probabilities (default 10.0).

-d dir

This specifies the directory to search for the HMM definition files corresponding to the labels used in the recognition network.

-e

When using direct audio input, output transcriptions are not normally saved. When this option is set, each output transcription is written to a file called PnS

where n is an integer which increments with each output file, P and S are strings which are by default empty but can be set using the configuration variables `RECOUTPREFIX` and `RECOUTSUFFIX`.

`-f`

During recognition keep track of full state alignment. The output label file will contain multiple levels. The first level will be the state number and the second will be the word name (not the output symbol).

`-g`

When using direct audio input, this option enables audio replay of each input utterance after it has been recognised.

`-h mask`

Set the mask for determining which transform names are to be used for the input transforms.

`-i s`

Output transcriptions to MLF s .

`-j i`

Perform incremental MLLR adaptation every i utterances

`-k`

Use an input transform (default off).

`-l dir`

This specifies the directory to store the output label files. If this option is not used then HVITE will store the label files in the same directory as the data. When output is directed to an MLF, this option can be used to add a path to each output file name. In particular, setting the option `-l '*'` will cause a label file named xxx to be prefixed by the pattern `"/xxx"` in the output MLF file. This is useful for generating MLFs which are independent of the location of the corresponding data files.

`-m`

During recognition keep track of model boundaries. The output label file will contain multiple levels. The first level will be the model number and the second will be the word name (not the output symbol).

`-n i [N]`

Use i tokens in each state to perform N -best recognition. The number of alternative output hypotheses N defaults to 1.

`-o s`

Choose how the output labels should be formatted. s is a string with certain letters (from NSCTWM) indicating binary flags that control formatting options. N normalise acoustic scores by dividing by the duration (in frames) of the segment.

S remove scores from output label. By default scores will be set to the total likelihood of the segment. C Set the transcription labels to start and end on frame centres. By default start times are set to the start time of the frame and end times are set to the end time of the frame. T Do not include times in output label files. W Do not include words in output label files when performing state or model alignment. M Do not include model names in output label files when performing state and model alignment.

-p f

Set the word insertion log probability to f (default 0.0).

-q s

Choose how the output lattice should be formatted. s is a string with certain letters (from ABtvaldmn) indicating binary flags that control formatting options. A attach word labels to arcs rather than nodes. B output lattices in binary for speed. t output node times. v output pronunciation information. a output acoustic likelihoods. l output language model likelihoods. d output word alignments (if available). m output within word alignment durations. n output within word alignment likelihoods.

-r f

Set the dictionary pronunciation probability scale factor to f. (default value 1.0).

-s f

Set the grammar scale factor to f. This factor post-multiplies the language model likelihoods from the word lattices. (default value 1.0).

-t f [i l]

Enable beam searching such that any model whose maximum log probability token falls more than f below the maximum for all models is deactivated. Setting f to 0.0 disables the beam search mechanism (default value 0.0). In alignment mode two extra parameters i and l can be specified. If the alignment fails at the initial pruning threshold f, then the threshold will be increased by i and the alignment will be retried. This procedure is repeated until the alignment succeeds or the threshold limit l is reached.

-u i

Set the maximum number of active models to i. Setting i to 0 disables this limit (default 0).

-v f

Enable word end pruning. Do not propagate tokens from word end nodes that fall more than f below the maximum word end likelihood. (default 0.0).

-w [s]

Perform recognition from word level networks. If *s* is included then use it to define the network used for every file.

-x ext

This sets the extension to use for HMM definition files to *ext*.

-y ext

This sets the extension for output label files to *ext* (default *rec*).

-z ext

Enable output of lattices (if performing NBest recognition) with extension *ext* (default off).

-L dir

This specifies the directory to find input label (when *-a* is specified) or network files (when *-w* is specified).

-X s

Set the extension for the input label or network files to be *s* (default value *lab*).

-E dir [ext]

Parent transform directory and optional extension for parent transforms. The default option is that no parent transform is used.

-G fmt

Set the label file format to *fmt*.

-H mmf

Load HMM macro model file *mmf*. This option may be repeated to load multiple MMFs.

-I mlf

This loads the master label file *mlf*. This option may be repeated to load several MLFs.

-J dir [ext]

Add directory to the list of possible input transform directories. Only one of the options can specify the extension to use for the input transforms.

-K dir [ext]

Output transform directory and optional extension for output transforms. The default option is that there is no output extension and the current transform directory is used.

-P fmt

Set the target label format to *fmt*.

Appendix B – Other Tools and Resources

SAAVB

Saudi Accented Arabic Voice Bank (SAAVB), developed by King Abdulaziz City for Science and Technology (KACST) and approved by IBM Egypt, is a speech corpus developed for phone-based speech recognition system.

SAAVB contains a corpus of speech waves and their transcriptions of more than 1000 speakers covering all the regions in Saudi Arabia with statistical distribution of region, age and gender. The corpus has more than 300,000 electronic files.

SAAVB is completely owned by KACST, and can be licensed to companies or research centers to train their engines to recognize Arabic speech.

ATK

The Application Toolkit for HTK (ATK) is an application programming interface (API) designed to facilitate building experimental applications for HTK. It consists of a C++ layer sitting on top of the standard HTK libraries. This allows novel recognizers built using customized versions of HTK to be compiled with ATK and then tested in working systems. Like HTK itself, it is portable across the main Unix platforms and Windows.

ATK features include:

- Multi-threaded to allow efficient and responsive real-time operation.
- Synchronised audio input/output with barge-in support.
- Support for finite-state grammars and trigram language models.
- Ability to return recognition results word-by-word as they are recognised to reduce latency
- N-best recognition output
- Support for HLDA
- Integrated Flite speech synthesis.
- Make files for single-build under Linux and Windows

Details on usage are explained in ATK Reference Manual (available online at CUED).

BNF

Knowledge representation languages consist of an infinite set of strings. A concise way to characterize the set is by means of a grammar. We write our grammar in a formalism called Backus-Naur form (BNF). There are four components to a BNF grammar:

- A set of terminal symbols. These are the symbols or words that make up the strings of the language. They could be letters (A, B, C ...) or words (a, aardvark, abacus ...).
- A set of non-terminal symbols that categorize sub-phrases of the language. For example, the non-terminal symbol *Nounphrase* in English denotes an infinite set of strings including "you" and "the big slobbery dog."
- A start symbol, which is the non-terminal symbol that denotes the complete strings of the language. In English, this is *Sentence*; for arithmetic, it might be *Expr*.
- A set of rewriter rules, of the form $LHS \rightarrow RHS$, where *LHS* is a non-terminal and *RHS* is a sequence of zero or more symbols (either terminal or non-terminal).

A rewrite rule of the form

$Sentence \rightarrow NounPhrase\ VerbPhrase$

means that whenever we have two strings categorized as a *NounPhrase* and a *VerbPhrase*, we can append them together and categorize the result as a sentence. As an abbreviation, the symbol | can be used to separate alternative right-hand sides. Here is a BNF grammar for simple arithmetic expressions:

$Expr \rightarrow Expr\ Operator\ Expr \mid (Expr) \mid Number$

$Number \rightarrow Digit \mid Number\ Digit$

$Digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$Operator \rightarrow + \mid - \mid / \mid *$

Other BNF notations exist; for example, <Digit> may be used instead of Digit for a non-terminal, 'word' instead of word for a terminal, or : = instead of \rightarrow in a rule.

Total Recorder

Total Recorder Developer Edition Tester's Version 6.1

Total Recorder is a powerful tool for recording, processing, converting, and playing sound. It can execute a command line after a file has been recorded or after a recording session has completed. This feature is available when recording in split mode and for a scheduling recording.

Roman Representations of Arabic Characters

Arabic Character	Roman Representation
ـَ (fat-ha)	a
ـُ (dhamma)	u
ـِ (kasra)	i
ـْ (sukoon)	0
ـ (shakhta)	1
ـّ (shadda)	2
ـّ (madd)	G
ـَ (tanween fat-h)	A
ـِ (tanween kasr)	I
ـُ (tanween dhamm)	U
ب	b
ث	B
ذ	c
ظ	C
د	d
ض	D
أ	E
ف	f
آ	F
هـ	h
ح	H
ج	j

Arabic Character	Roman Representation
ق	K
ل	L
إ	L
م	m
و	M
ن	n
ئ	N
ة	O
ي	P
ا	q
ء	Q
ر	r
ع	R
س	s
ص	S
ت	t
ط	T
و	w
غ	x
خ	X
ي	Y
ز	z

Arabic Character	Roman Representation
ك	k

Arabic Character	Roman Representation
ش	z

Phonetic Representation of Arabic Sounds

Arabic contains 34 sounds; 28 consonants, 3 short vowels (fat-ha, dhamma, and kasra), and 3 long vowels (madd alef, madd waw, and madd yaa). “Sukoon” is the absence of sound and “Shadda” is the repetition of the sound.

Arabic Sound	Phonetic Symbol
ـَ (fat-ha)	al
ب	bl
ث	bu
ذ	cl
ظ	cu
د	dl
ض	du
ف	fl
ـِ (kasra)	il
ج	jl
ك	kl
ق	ku
ل	ll
م	ml
ن	nl

Arabic Sound	Phonetic Symbol
ـْ (madd alef)	ql
ء	qu
ر	rl
ع	ru
س	sl
ص	su
ت	tl
ط	tu
ـُ (dhamma)	ul
و (madd waw)	wl
و	wu
غ	xl
خ	xu
ي (madd yaa)	yl
ي	yu
ز	zl
ش	zu

Appendix C – Our Steps

Formal Testing of Accuracy Rates

Strategy 1

Prototype1	
Mono- phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Wed May 23 02:51:50 2007 Ref : Data/Word_LB Rec : Output\TEST.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=69, N=69] WORD: %Corr=51.42, Acc=40.66 [H=325, D=41, S=266, I=68, N=632] ===== </pre>
Mono- phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 14:25:43 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMonoBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=67, N=67] WORD: %Corr=47.15, Acc=28.94 [H=413, D=42, S=160, I=235, N=615] ===== </pre>
Tri-phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Sat May 26 14:39:17 2007 Ref : MLF/MLFword.mlf Rec : test.mlf ----- Overall Results ----- SENT: %Correct=29.85 [H=20, S=47, N=67] WORD: %Corr=62.76, Acc=59.84 [H=509, D=11, S=95, I=18, N=615] ===== </pre>
Tri-phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 14:16:48 2007 Ref : MLF/MLFword.mlf Rec : Output/TestTriBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=67, N=67] WORD: %Corr=61.93, Acc=46.50 [H=510, D=10, S=95, I=224, N=615] ===== </pre>

Prototype2	
Mono- phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 15:20:09 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMono.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=72, N=72] WORD: %Corr=46.59, Acc=33.69 [H=307, D=11, S=341, I=85, N=659] ===== </pre>
Mono- phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 21:04:05 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMonoBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=72, N=72] WORD: %Corr=89.23, Acc=54.93 [H=588, D=8, S=63, I=226, N=659] ===== </pre>
Tri-phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Wed May 23 09:13:58 2007 Ref : Data/Word_LB Rec : Output\output.mlf ----- Overall Results ----- SENT: %Correct=11.11 [H=8, S=64, N=72] WORD: %Corr=78.30, Acc=75.87 [H=516, D=3, S=140, I=16, N=659] ===== </pre>
Tri-phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 22:06:39 2007 Ref : MLF/MLFword.mlf Rec : Output/TestTriBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=72, N=72] WORD: %Corr=98.94, Acc=76.02 [H=652, D=1, S=6, I=151, N=659] ===== </pre>

Prototype3	
Mono- phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 14:50:32 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMono.mlf ----- Overall Results ----- SENT: %Correct=1.37 [H=1, S=72, N=73] WORD: %Corr=50.98, Acc=36.65 [H=338, D=71, S=254, I=95, N=663] ===== </pre>

Prototype3	
Mono- phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 14:47:42 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMonoBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=73, N=73] WORD: %Corr=59.58, Acc=20.21 [H=395, D=42, S=226, I=261, N=663] ===== </pre>
Tri-phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Thu May 24 15:20:01 2007 Ref : MLF/MLFword.mlf Rec : Output/RecResult.mlf ----- Overall Results ----- SENT: %Correct=5.48 [H=4, S=69, N=73] WORD: %Corr=67.57, Acc=57.47 [H=448, D=39, S=176, I=67, N=663] ===== </pre>
Tri-phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Sun May 27 15:12:00 2007 Ref : MLF/MLFword.mlf Rec : Output/TestTriBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=73, N=73] WORD: %Corr=66.82, Acc=26.70 [H=443, D=35, S=185, I=266, N=663] ===== </pre>

Strategy 2

Recorded System	
Mono- phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Tue May 29 22:36:05 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMono.mlf ----- Overall Results ----- SENT: %Correct=1.47 [H=1, S=67, N=68] WORD: %Corr=57.60, Acc=55.36 [H=360, D=39, S=226, I=14, N=625] ===== </pre>
Mono- phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Tue May 29 23:03:26 2007 Ref : MLF/MLFword.mlf Rec : Output/TestMonoBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=68, N=68] WORD: %Corr=88.64, Acc=64.96 [H=554, D=0, S=71, I=148, N=625] ===== </pre>

Recorded System	
Tri-phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Wed May 30 01:23:27 2007 Ref : MLF/MLFword.mlf Rec : Output/TestTri.mlf ----- Overall Results ----- SENT: %Correct=4.41 [H=3, S=65, N=68] WORD: %Corr=74.08, Acc=72.96 [H=463, D=32, S=130, I=7, N=625] ===== </pre>
Tri-phones + bi-grams	<pre> ===== HTK Results Analysis ===== Date: Wed May 30 01:26:37 2007 Ref : MLF/MLFword.mlf Rec : Output/TestTriBi.mlf ----- Overall Results ----- SENT: %Correct=0.00 [H=0, S=68, N=68] WORD: %Corr=87.92, Acc=74.56 [H=612, D=0, S=13, I=146, N=625] ===== </pre>

Strategy 3

AraDict	
Mono- phones + word loop	<pre> ===== HTK Results Analysis ===== Date: Sat May 26 14:08:45 2007 Ref : MLF/MLFword.mlf Rec : test.mlf ----- Overall Results ----- SENT: %Correct=10.00 [H=1, S=9, N=10] WORD: %Corr=93.33, Acc=89.20 [H=56, D=8, S=6, I=2, N=60] ===== </pre>

Files Created Manually Before Training

- A configurations file (config.conf) to be used throughout the training process.

```

# Feature Configuration
TARGETKIND = MFCC_0_D_A
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = T

```

```
# Source File Format
SOURCEFORMAT = WAV
SOURCERATE = 454.54
```

- A conversion list script file (ConverList.scp) to be used with HCopy. It contains the name of each wave file and the corresponding mfcc file name needed to be created.

```
Data\wav\10150401.wav      Data\wav\10150401.mfcc
Data\wav\10150402.wav      Data\wav\10150402.mfcc
Data\wav\10150403.wav      Data\wav\10150403.mfcc
Data\wav\10150404.wav      Data\wav\10150404.mfcc
Data\wav\10150405.wav      Data\wav\10150405.mfcc
Data\wav\10150406.wav      Data\wav\10150406.mfcc
Data\wav\10150408.wav      Data\wav\10150408.mfcc
...                          ...
```

- A training list script file (TrainList.scp) to be used throughout the training process. It contains the list of mfcc files.

```
Data\mfcc\10150401.mfcc
Data\mfcc\10150402.mfcc
Data\mfcc\10150403.mfcc
Data\mfcc\10150404.mfcc
Data\mfcc\10150405.mfcc
...
```

- A model prototype (Proto.txt)

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "Proto"
<BeginHMM>
  <NumStates> 5
    <state> 2
      <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
      <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
    <state> 3
      <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
      <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

```

        <state> 4
            <Mean> 39
            0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            <Variance> 39
            1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
        <TransP> 5
        0.0 1.0 0.0 0.0 0.0
        0.0 0.6 0.4 0.0 0.0
        0.0 0.0 0.6 0.4 0.0
        0.0 0.0 0.0 0.7 0.3
        0.0 0.0 0.0 0.0 0.0
    <EndHMM>

```

- A monophone dictionary (dictionary.txt)

```

BlqBh bu ll ql bu hl sp
BlqBmyh bu ll ql bu ml yl hl sp
BlqByn bu ll ql bu yl nl sp
Bmqnyh bu ml ql nl yl hl sp
Chrqn cu hl rl ql nl sp
EHcf qu hu cl fl sp

```

- An hmm List (hmmlist.txt)

```

Sil
sp
al
bl
bu
cl

```

- A unique sorted word list (WordList.txt)

```

BlqBh
BlqBmyh
BlqByn
Bmqnyh
Chrqn
EHcf
Ehlqm
EHmd
ERd
EZjqn

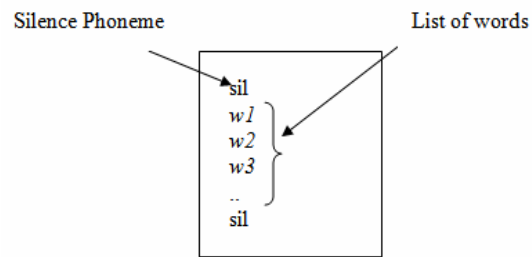
```

- Command files

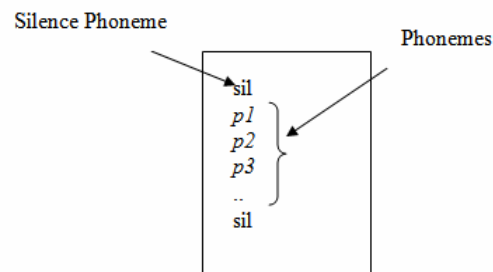
File Name	Content
co.hed	(Empty)
tri.hed	WB sp WB sil TC
sil.hed	AT 2 4 0.2 {sil.transP} AT 4 2 0.2 {sil.transP} AT 1 3 0.3 {sp.transP} TI silst {sil.state[3],sp.state[2]}
mmftri.hed	CL Lists/tri-phones.txt TI T_al {(*-al+*, al+*, *- al).transP} TI T_bl {(*-bl+*, bl+*, *- bl).transP} TI T_bu {(*-bu+*, bu+*, *- bu).transP} TI T_cl {(*-cl+*, cl+*, *- cl).transP} TI T_cu {(*-cu+*, cu+*, *- cu).transP} TI T_dl {(*-dl+*, dl+*, *- dl).transP} TI T_du {(*-du+*, du+*, *- du).transP} TI T_fl {(*-fl+*, fl+*, *- fl).transP} TI T_hl {(*-hl+*, hl+*, *- hl).transP} TI T_hu {(*-hu+*, hu+*, *- hu).transP} TI T_jl {(*-il+*, il+*, *- il).transP} TI T_jl {(*-jl+*, jl+*, *- jl).transP} TI T_kl {(*-kl+*, kl+*, *- kl).transP} TI T_ku {(*-ku+*, ku+*, *- ku).transP} TI T_ll {(*-ll+*, ll+*, *- ll).transP} TI T_ml {(*-ml+*, ml+*, *- ml).transP} TI T_nl {(*-nl+*, nl+*, *- nl).transP} TI T_ql {(*-ql+*, ql+*, *- ql).transP} TI T_qu {(*-qu+*, qu+*, *- qu).transP}

	<pre> TI T_rl {(*-rl+*, rl+*, *- rl).transP} TI T_ru {(*-ru+*, ru+*, *- ru).transP} TI T_sl {(*-sl+*, sl+*, *- sl).transP} TI T_su {(*-su+*, su+*, *- su).transP} TI T_tl {(*-tl+*, tl+*, *- tl).transP} TI T_tu {(*-tu+*, tu+*, *- tu).transP} TI T_uu {(*-ul+*, ul+*, *- ul).transP} TI T_wl {(*-wl+*, wl+*, *- wl).transP} TI T_wu {(*-wu+*, wu+*, *- wu).transP} TI T_xl {(*-xl+*, xl+*, *- xl).transP} TI T_xu {(*-xu+*, xu+*, *- xu).transP} TI T_yl {(*-yl+*, yl+*, *- yl).transP} TI T_yu {(*-yu+*, yu+*, *- yu).transP} TI T_zl {(*-zl+*, zl+*, *- zl).transP} TI T_zu {(*-zu+*, zu+*, *- zu).transP} TI T_sil {(*-sil+*, sil+*, *- sil).transP} TI T_sp {(*-sp+*, sp+*, *- sp).transP} </pre>
<pre> mixX.hed (Where is X is a number from 2 to 20) </pre>	<pre> MU ? {*.state[2-4].mix} </pre>

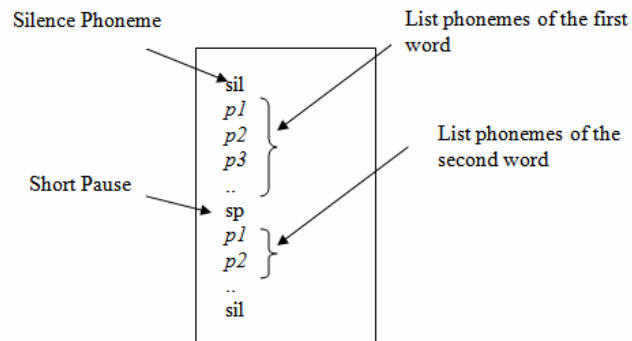
Label Files



Word based label file format

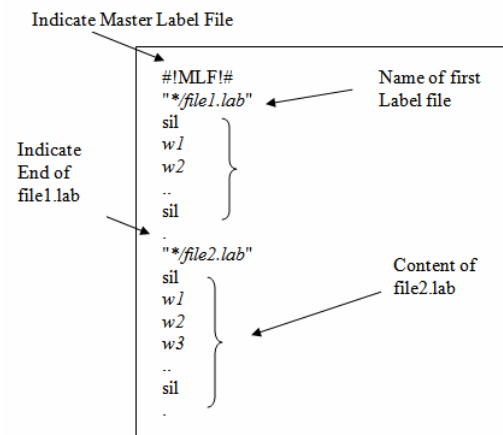


Phonetic Based Label File Format without short pause

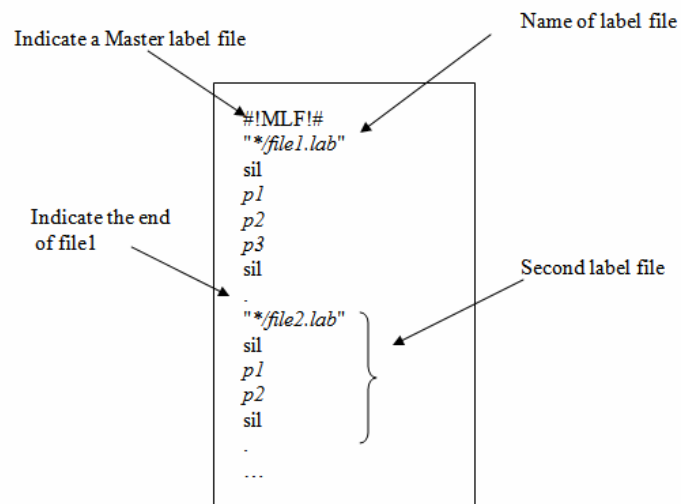


Phonetic Based Label File Format with short pause

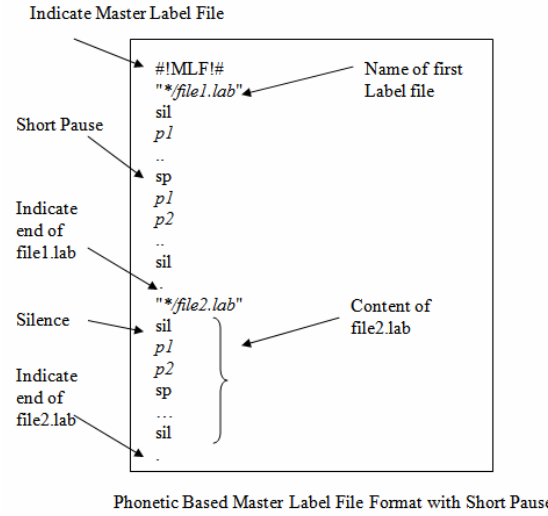
MLFs



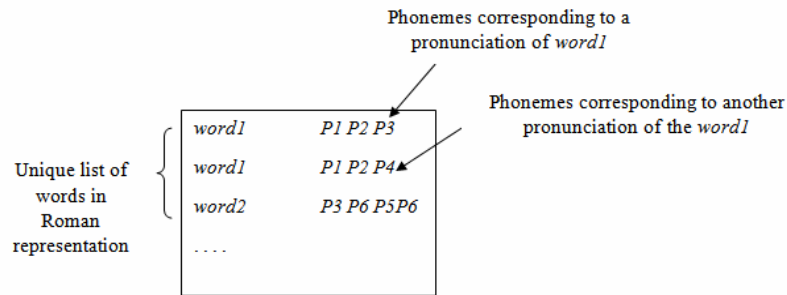
Word Based Master Label File Format



Phonetic Based Master Label File Format
without short pause



Pronunciation Dictionary



Complete List of Words in AraDict Dictionary

زين	صحه	انتقل
سارة	عال	صفر
سعيد	عرب	واحد
شدى	فيز	اثنين
شروق	نجم	ثلاثة
صالح	أسيل	أربعة
ضفاف	أميرة	خمسة
عبدالله	ابراهيم	ستة
عمر	المقرن	سبعة

علي	السلامة	ثمانية
عشقي	العابدين	تسعة
محمد	الغامدي	احص
مريم	الثنيان	تال
نوح	خلود	حسب
نوف	رادين	دار
نورة	رزقي	ريض
هيفاء	رادين	سلم

Application Functions

The following functions were implemented as part of AraDict:

	Function Name	Description
1.	ImageToPictureDisp	Converts the image type to match the one required by the ribbon.
2.	GetImage	Displays the image onto the buttons of the ribbon.
3.	Connect	Constructor for the add-in application. Sets the start up path.
4.	OnConnection	Specifies that the hosting application will be MS Word 2007 and loads the add-in application.
5.	New_User	Call back function for the “add new user” button. Starts up the new user creation wizard.
6.	GetLabel	Changes the label of the stop/print button according to the recognition method selected (live/recorded)
7.	GetPrintImage	Changes the picture of the stop/print button according to the recognition method selected (live/recorded)
8.	GetText	Responsible for setting the status in the status box
9.	Template	Call back function for the “templates” drop down list. Opens a new template according to selection.

	Function Name	Description
10.	User	Call back function for the “users” drop down list. Sets the user path to the user folder according to selection.
11.	Live	Call back function for the “live” button. Sets the recognition to live mode. Ensures user has been first selected. Calls <code>Run_HVite_Live</code> .
12.	Recorder	Call back function for the “recorder” button. Sets the recognition to recorded mode. Ensures user has been first selected. Calls <code>Run_Recorder</code> , <code>Run_HVite_Recorded</code> and <code>Romantoarabic</code> functions.
13.	OnAction	Print function for both <code>Live</code> and <code>Recorder</code> functions.
14.	ribbonLoaded	Dynamically updates the ribbon after it has been loaded.
15.	Run_Recorded_Bat	Calls a batch files that in turn runs a set of batch files that call <code>TotalRecorder</code> , <code>HVite</code> and <code>RomanToArabic</code> for the recorded file.
16.	Run_HVite_Live	Runs <code>HVite-live</code> batch file to run the <code>HVite</code> tool with the live configuration file
17.	Romantoarabic	Call the <code>Romantoarabic</code> batch file and send as an argument the file name that will be converted from Roman to Arabic
18.	Print_Temp1	Prints an array of characters of a predefined size into template one
19.	Print_Temp2	Prints an array of characters of a predefined size into template two
20.	Print_Temp3	Prints an array of characters of a predefined size into template three
21.	Print	Prints the content of a specific file into an empty word document
22.	nextcell1	Moves the curser into the next cell in Template One
23.	nextcell2	Moves the curser into the next cell in Template two
24.	nextcell3	Moves the curser into the next cell in Template three

Appendix D – External Communications Log

4/Feb/2007 9:24 PM	Dr. Mansour AlGhamdi, KACST
Description	<ul style="list-style-type: none"> Described the project and its purpose Requested a speech corpus, for the purpose of training the ASR system, with the following characteristics: noiseless speeches of phonetically rich words, phonetically rich sentences, yes/no, digits, numbers, currencies, dates, times, computer commands, alphabets, months, days, and seasons, as recommended to us by our supervisor
Outcome	Granted a small sample of the Saudi Accented Arabic Voice Bank (SAAVB) by the end of Feb.

20/Feb/2007 10:56 AM	Ossama Emam, Manager, Human Language Technologies Group, IBM Egypt Branch
Description	<ul style="list-style-type: none"> Described the project and its purpose Requested a speech corpus Asked for advice regarding implementing an ASR system
Outcome	<ul style="list-style-type: none"> Advised us against implementing a “real-time” dictation system Recommended developing an offline, small vocabulary Arabic speech recognition system with a small corpus that we should record locally Said he could not provide us with a speech corpus due to legal issues

19/Apr/2007 12:33 PM	Professor Steve Young, Head of the Information Engineering Division at the University of Cambridge
Description	<ul style="list-style-type: none"> • Described the project and its purpose • Expressed problems faced with ATK version 1.4.1 • Enquired about version 1.5 that was supposed be released in 2005 but couldn't be found on the website
Outcome	<ul style="list-style-type: none"> • Responded saying version 1.5 was never released • Sent us a Beta version of ATK 1.6 for Windows after 5 days

Apr/2007 (phone)	Dr. Mansour AlGhamdi, KACST
Description	<ul style="list-style-type: none"> • Asked if he was familiar with HTK
Outcome	<ul style="list-style-type: none"> • Referred us to Ammar Al-Anazi

Apr/2007 (phone)	Ammar Al-Anazi, KACST
Description	<ul style="list-style-type: none"> • Described the project and its purpose • Described voice bank and tools • Described our progress so far • Described problems faced with live recognition • Asked if any suggestions could be made
Outcome	<ul style="list-style-type: none"> • Confirmed his familiarity with HTK • Confirmed his familiarity with SAAVB • Pointed out a mismatch in sampling rates between training data (SAAVB mobile recorded) and new data (obtained from microphone) • Suggested using offline recognition instead of live recognition • Suggested using GoldWave software, which enables us to record using our choice of sampling rate • Requested to see our configuration file

1/May/2007 11:45 AM	Ammar Al-Anazi, KACST
Description	<ul style="list-style-type: none"> Responded to his request and sent him our configuration file
Outcome	<ul style="list-style-type: none"> Suggested an equation to calculate the sampling rate Suggested manually adjusting the settings of the configuration files using the result of the equation Pointed out that there are no standard sampling rates for recording new files on the computer and that they are machine dependent

1/May/2007 2:09 PM	Ammar Al-Anazi, KACST
Description	<ul style="list-style-type: none"> Informed him of the updates: both solutions were tested but neither worked (GoldWave and adjusting the configuration file) Asked if the problem could lie within the training process itself
Outcome	<ul style="list-style-type: none"> Asked to see entire training process

1/May/2007 4:36 PM	Dr. Jalal Al-Muhtadi, CCIS, KSU
Description	<ul style="list-style-type: none"> Described the project and its purpose Described voice bank and tools Described problems faced with recognition Asked if any suggestions could be made
Outcome	<ul style="list-style-type: none"> Responded saying he was unfamiliar with the tools

2/May/2007 1:23 PM	Professor Steve Young, Head of the Information Engineering Division at the University of Cambridge
Description	<ul style="list-style-type: none"> • Described the problem of poor recognition • Enquired about whether or not it is possible to use one sampling rate for training (mobile device) and a different sampling rate for recognition (microphone)
Outcome	<ul style="list-style-type: none"> • Responded saying both sampling rates must be identical to avoid shifting frequencies and mismatch of acoustic models

14 Glossary

Accuracy rate

Rate of sensing and conversion of data into machine-readable form by a computer

Acoustic

Relating to sound, the sense of hearing, or the science of sound

Coarticulation

The influence of the sound of one word with the sound of the next word

Constant environment

An environment where the disturbance is relatively stable

Continuous Speech

The continuous utterance of words, requiring no pause between words

Diacritic

A mark above or below a printed letter that indicates a change in the way it is to be pronounced or stressed

Dictation

The act of dictating a text or letter, or of writing down what is being dictated

Dictionary

A pronunciation dictionary contains a list of words and the sequence of phonemes corresponding to the pronunciation of the word

Discrete

Describes mathematical elements or variables that are distinct, unrelated, and have a finite number of values

Disfluency

A disruption in the smooth flow or expression of speech

Extralinguistic

Not included within the realm of language

Feature Extraction

The process of segmenting a signal into successive frames to extract vectors of acoustical coefficients

Hidden Markov Model (HMM)

A statistical model where the system being modelled is assumed to be a process with unknown parameters, and the challenge is to determine the hidden parameters from the observable parameters

Isolated Speech

The utterance of a single word at a time and requiring a pause between saying each word

Latency

Time delay between the moment speech is initiated by the speaker and the moment results becomes detectable

Linguistic

Relating to language or languages

Noise

A loud, surprising, irritating, or unwanted sound

Out of Vocabulary Word (OOV)

A word not existing within a speech recognition system dictionary

Phoneme

A speech sound that distinguishes one word from another, e.g. the sounds "d" and "t" in the words "bid" and "bit." A phoneme is the smallest phonetic unit that can carry meaning

Phonetic

The system or pattern of speech sounds used in a language

Real Time Factor (RTF)

The ability of the system to respond in a real time manner

Signal

Information transmitted by means of a modulated current or an electromagnetic wave and received by telephone, telegraph, radio, television, or radar

Signal to noise ration (SNR)

Measured in decibels, this is the difference between the signal strength a system reproduces compared to the strength or amplitude of its background noise

Speaker Adaptation

The concept of training the system to the characteristics of a new speaker in order to increase the accuracy rate

Speaker Dependent

Systems which rely on an individual speaker's voice characteristics in order to successfully process speech

Speaker Independent

Systems designed for a variety of speakers without the need to be trained by the user

Speech Corpus

A speech corpus is a collection of recorded utterances and their associated transcriptions used as a basis for the descriptive analysis of a language

Speech Recognition

A system of computer input and control in which the computer can recognize spoken words and transform them into digitized commands or text. With such a system, a computer can be activated and controlled by voice commands or take dictation as input to a word processor or a desktop publishing system

Syllable

A physical unit of organization for a sequence of speech sounds

Training

The process of mapping speech features to phonemes in order to obtain the model parameters of a HMM from a set of example data

Usability

Systems ease-of-use, clarity of the interface, and availability of a proper user manual

Vowel

A speech sound produced by the passage of air through the vocal tract, with relatively little obstruction

Word Error Rate (WER)

The percentage of incorrectly recognized words

15 Expressions of Gratitude

“Read, for your Lord is most gracious, Who has taught (you) by the pen”
(Qur'an 96:3-4)

“First and for most I am grateful to Allah, the most Gracious the most Merciful, for granting me the health, the opportunity, and the ability to pursue my education, and to have a wonderful helpful family and teachers who guided me for better education and better life.

I am thankful to the wonderful time I spent with my group project team, AraDiet Team: Aciel Eshki, Dhefah Radain, Haifa AlThonayan, Kholoud Zain AlAbdeen, Mariam Nouh, and Sheroug AlMegren. They were and they are wonderful colleagues, class mates, and friends. We shared the work and they were helpful and honest in their valuable advice and dedication for first class achievement in our project.

All of that were on the expense of the time and worry of my wonderful mother, Laila Khadr, who will stay awake all night beside me to give the moral and spiritual support as well as her valuable academic experience.

I am specially grateful to my father, Abdullah Al-Salamah, who made our hour hour technology driven by having abundant broadband Internet and having it all wireless so as to make knowledge seeking effort enjoyable.

I am also thankful to my brothers and sisters who were of great help. In particularly, my dearest sister, Hessah, and my brother, Abdulrahman, for their valuable assistance and academic advice.

Special thanks to Amal Al-Salamah and Sahar Othman for their enduring support and encouragement.

I would love at the end to thank my colleagues in King Saud University, relatives, and all other people for their loving support at various stages of this project.

Lastly, I ask of the superiority of Allah to accept this humble work, guide our steps and grant us the way to what pleases Him”

-Shada

“I stand speechless before those who have contributed to my life.

I thank God for constantly answering my prayers no matter how unattainable they seem, for always guiding me towards a better decision than my own, for always placing me in the right place at the right time, and for blessing me with great abilities, that I will gladly use to serve Him.

I thank God for blessing me with such a wonderful family; a caring mother who has devoted her life to her children in every possible way; a father who through laughter and wisdom lit our way; three brothers so reliable and dependable that it strengthens me everyday; and my little sister, Anoof, joy of life, I can't believe how much you've grown; I still dream of you as a little girl when I sleep.

I thank god for placing such amazing people along my path:

My lifelong partner and true best friend, Saeed; you are my strangest nerdy male equivalent! (or opposite, depending on how you look at it) You baffle me with your strange mixture of... everything! I love how you make me laugh; I love your endless support and undeniable kindness.

Mrs. Samira Lahmar, who, in the first two years of college, had opened my eyes to the magic of algorithms and programming. You never compromised on your high standards and your strict approach lead me to discover my true passion. You showed me that the word “genius” is attainable. Thank you for introducing me to the beautiful world of computing!

My team members, who showed me that friendship and acceptance is more valuable than strict deadlines. You were all so kind during my pregnancy; the simplest gestures like placing a pillow behind my back while we worked meant the world to me. You are the best group anyone can ask for!”

-Aciel

“I would like to offer special thanks to my loved husband who has supported me with everything I went through all the years we were together. His patience was a major factor that helped me to continue the good work on this project. A lot of times he was the reason that made me challenge myself to come out with the best thing I can in different aspects of my life. I wish all the best for him.

A big hug to the best mother in the world, my mom. She has always been an inspiration to me throughout my life. She has supported me in all the good things I have done, and advised me about the bad things. Her prayers helped me a lot in every aspect of my life. Thank you mom.

My Dad, he is the kindest father any one can ever have. I thank my god every day that he is my dad. He was the main person who encouraged me to be successful. I would like to thank him for his friendship, wisdom and understanding.”

-Dhefaf

“To my dear parents for their ultimate support and for positive encouragement and above of all for their faithful prayer.

Grateful thanks to dear Miss. Majda Wazzan for her endless help and warm feeling towards me.

I ask God to reward all of those who stood by me in the hard time & gave me from their time just to cheer me up.”

-Kholoud

“I dedicate this project to my family especially my mum and dad who managed to support me throughout this experience.

I would also like to dedicate this to my grandfather who taught me how to persevere and overcome obstacles with confidence.

Also my friends, who put up with my explosive mood swings with humour.

Last but not least, my Barney-addicted nephews, if not for their addiction I wouldn't have found the project as a way to escape.”

-Sheroug

“First and foremost, I'm thankful for Allah as without whose blessings nothing is possible.

I would like to give a special and great thanks to my father and my mother for their great support and encouragement that helped me complete this project.

I'm thankful also for all my brothers and sisters Thamer, Amr, Mohammed, Lina and my lovely sister Emy and her husband Nizar for their constant support and enthusiasm.

Thanks also to my cousin Samah for her caring, concern and attention.

Finally, special thanks to the AraDict Team: It was a pleasure working with you girls on this project, you all put an amazing effort into it.”

-Mariam

“I am thankful for my family especially my mother and my sister Atedal for their encouragement and their support during all the hard work in this project.

I would also like to thank my friends who have been with me through the whole project from day one till the end. Wish you all of the success in your life girls.

And special thanks to Sara, Haifa for their delightful help.”

-Haifa